VECTOR

144 pages of Array Programming

The Journal of the British APL Association

A Specialist Group of the British Computer Society

ISSN 0955-1433
www.vector.org.uk
Vol.14 No.3 January 1998

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+Win, IBM APL2/PC or Dyalog APL/W, or in documents from Windows Write (use the APL2741 TrueType font, available free from Vector Production), and MS Word.

Membership Rates 1997-98

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 Email: apl385@compuserve.com

Contents

J + OOP = J Release 4

J software Release 4 from Iverson Software promises to be the best ever. This version brings true Object Oriented Programming to J, making it much easier to write large and complex systems. Extensions to locales include hierarchical paths and object locales that can be dynamically created and referenced. This allows you to extend and customize a host of built-in facilities such as the plot and grid objects.

The new grid object implements a fully-customizable spreadsheet that is ideal for data entry and formatting.

We have also improved the development environment to simplify working with projects and creating standalone systems.

Many new Labs, have been added to get you up and running quicker. We have also added the ability to create a LAB in RTF format, for example, to include standard math symbols.

The documentation has been completely revised for this release. It will be available in printed form, and in Windows Help file format.

Watch for the Release of J for Windows CE!

www.jsoftware.com

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

Editorial

by Ajay Askoolum

APL. This acronym either evokes a sense of comfort or one of confusion. When confused (state of lacking information), the typical reaction is 'What is APL?'.

Indeed, what is APL? It is a commercial language which is available on all platforms, from mainframes, MACs, Unix to Personal Computers running DOS, OS/2, Windows 3.1x, 95 and NT (including some shareware versions). In the infancy of personal computers, APL interpreters were available for CP/M machines too. APL is some 30 years old, and has offered a core level of consistency which can only be envied by other major programming languages which have evolved.

Today's APL strives for compatibility with IBM's APL2, irrespective of operating system platform. APL2 is a mainframe interpreter. However, the PC offerings, notably APL+Win from APL2000, Dyalog APL/W from Dyadic and APL.68000 from MicroAPL offer 'core' language compatibility with APL2 as well as Windows (or MAC in the case of APL.68000) established standards. IBM's Windows 95/NT version of APL2 is in Beta test level 7; the OS/2 version has been available for a long time.

Therefore, APL, irrespective of its tokens or specialised characters is now DDE, ODBC and Windows API compliant. APL can have a conversation with Microsoft Access, Word, Excel, Lotus123 (to name but a few) and can use third party add-ons such as FarPoint's VBX (Spread) and Visual Components ActiveX OCX (Formula One) with remarkable ease. TrueType fonts have eliminated the frustration of communicating written APL. APL had maturity from the start, in spite of the adversity of slow CPU, limited memory, storage and expensive hardware such as custom printer heads and character set chips; it has now come of age and works seamlessly, without requiring specialist hardware.

With this level of integration, APL is neither the 'stick' to be confronted with, nor is it the fledgling of a protective species which requires a protective environment. Why is it not more successful? More visible? It would be easy to suggest that this is entirely due to not having Microsoft as a benefactor. I can put forward three alternative explanations.

> First, is the APL publicity organ (the marketing of APL vendors/distributors as well as the numerous international APL journals, including *Vector*) in the wrong hands? Possibly so, because this is an agent preaching to the converted.

If not so, perhaps the medium of communication is not on a wavelength that reaches the wider world. Indeed the standard clichés that 'APL works with you, not against you' or 'APL takes one tenth of any other language, namely, volume of code and time to develop it' are only meaningful to the converted.

Second, APL is still in the primitive age when the written word had not been invented. APL users have vendors' manuals to hand, usually. The uninitiated will not come across APL anywhere else or when they do, they tend to read the epitaph of APL. On the other hand, a novice is spoilt for choice when attempting to find third party reference on any aspect of the world of computing, except APL. An APL user either no longer needs any reference, has become stagnant for the lack of it or knows exactly where to find it through his/her specialist network of contacts, such as Vector's product guide. Now I can use all the methods, properties and events of Formula One OCX version 5.0 but first I had to find out how to by guesswork: its documentation does not refer to APL, its help desk does not support APL per se and APL2000's documentation of it could not be more sparse. A viable alternative to finding out is to give up ...

Thirdly, the inherent nature of APL is suicidal. It solves problems very quickly and therefore its requirements such as mainframe CPU, which can have a year's lead time for planning and budgeting, are more immediate. A longer time cycle can be beneficial because it does not create a 'crisis'; it allows for a more leisurely approach to planning and development. But, the immediacy of APL ought to be an asset not a liability. I suspect that it is not the development cycle but poor APL coding which is the real culprit. APL is the least prescriptive language I know and it is completely oblivious of programmer abuse. Given that most APL programmers have crafted their skill with a great deal of personal effort (because you have to do everything, you learn everything), it is easy to understand not only the individual style of APL systems in existence but also the political naivety of APL professional in the midst of computing professionals at large. On balance, APL tends to lose the battle for survival, primarily because a life span of 30 years has been long enough to confirm a sense of recurring crisis with the maintenance of APL systems.

Where Next for APL?

The Vendors' concession to control structures or keywords is a major step forward not only for programmer productivity but also for easing the introduction of APL to other programmers. Core compatibility with APL2 also allows programmers to move from one interpreter to another but this is severely jeopardised by departures from the 'core' APL2 standards; the different quadAVs, IBM's own auxiliary processors, Dyadic's namespaces and the myriad of APL2000's quad functions are very distinct solutions to standard recurring problems. The result? One APL can talk to another APL only via an interpreter!

From the point of view of attracting new members to the APL fold, a major leap of faith is required of APL vendors: tutorial manuals. These should provide worked examples and commentary on realistic applications using today's APL. The objective would not be to teach but to illustrate techniques. A stock control or book club application with a GUI front end or indeed other standard exercises on programming courses would be suitable. Demonstration workspaces such as APL+Win's REL20 are fine for illustrating the power of APL but they make no effort in teaching how to harness the same power (reverse engineering is not a recommended alternative to tutorials).

Vector and other journals need a re-juvenated approach. This can only come from new people getting involved in writing for the journals, or by simply taking the way forward. APL needs to be seen, seen not just by decision makers but especially by passers by. Quite literally, we need to enable APL material to fall into the hands of casual browers in bookshops, newsagents etc. The emphasis needs to be on teaching good standards and promoting today's basic APL where nested arrays and Windows GUI are no longer specialist topics but integral to the language.

In future, Vector plans to organise issues by themes such as 'Windows Programming With APL' , 'APL Basics' and 'Sustainable APL Programming Standards'. Additionally, Vector will organise a number of tutorial/demonstration sessions (see APL News). This requires your contribution, in fact it requires the wider participation of the international APL community. Vector will endeavour to make good APL material accessible to as wide an audience as possible, has considered electronic distribution of material and has a web site for this and other purposes (www.vector.org.uk). Your individual contribution is a vital requirement for the continued success of APL. It is imperative to share/publicise APL insight and expertise, and that implies that the culture for APL development must evolve from being driven by keen individuals to being team based. That is, APL should evolve from the 'Here's the solution, what is the problem?' ideology to the 'For this problem, APL is demonstrably the effective solution' ideology.

Finally, an incentive: Vector will give one year's complimentary subscription for each article which is accepted for and published in a future issue. The forthcoming AGM is an ideal opportunity to express your ideas for Vector and the Association; please avail yourself of the opportunity and come forward.

A World of Opportunities...

Reuters Information Services in Toronto builds and operates one of the largest historical databases in the world. It serves the Financial industry through a variety of Reuters delivery products. Our Toronto unit has recently been overwhelmed with new requirements to enhance our offerings, while undertaking an effort to organise and catalogue our systems in preparation for Y2000 testing.

Programmer/Analysts

Full time positions will involve many aspects of the software development life cycle, including writing functional specifications, coding, testing and providing support. The focus will be on the maintenance of complex database systems, and will involve both applications and systems programming. You need two years of programming experience plus a degree in Computer Science, Computer Engineering, or Mathematics, or equivalent industry experience. You enjoy, and are challenged by, the prospect of solving complex problems, and your communication skills enhance your ability to work effectively in a team environment. APL experience, database management and knowledge of the financial data and information industry would be assets. If you are interested in full time opportunities, respond quoting file C0918.

Consulting positions are also available at Reuters, and this is one of the best times to become involved! Learn about the entire system and make yourself an asset for future contracts too. If you already know the Reuters applications then we want to hear from you. Considerations will be made for remote or on-site consulting. Strong APL skills (ideally Sharp APL) and experience are a must. Assignments can often be tailored to challenge and take advantage of the consultants experience and interests. If you are considering consulting opportunities, respond quoting file D0918.

These are superb career opportunities with an industry leader that recognizes achievement in very tangible ways. To apply, please send your résumé to:

Human Resources
Reuters Information Services (Canada) Limited
The Exchange Tower, Suite 2000
Toronto, Ontario M5X 1E3
Fax: (416) 364-9017

We appreciate your interest and will contact you if a meeting is appropriate.
REUTERS is an equal opportunity employer.

Quick Reference Diary 1998–1999

Date	Venue	Event
May 22nd	RSS, London	BAA Vendor Forum and AGM
July 27-31	Rome	APL98
August 10-14 1999	Scranton, Penn.	APL99

Dates for Future Issues of VECTOR

	Vol.14 No.4	Vol.15 No.1	Vol.15 No.2
Copy date	10th April	3rd July	4th Sept
Ad booking	17th April	10th July	11th Sept
Ad Copy	24th April	19th July	18th Sept
Distribution	May 98	August 98	October 98
Edited by	Stefano Lanzavecchia	Anthony Camacho	Alan Sykes & Alan Mayer

Vector Back Numbers

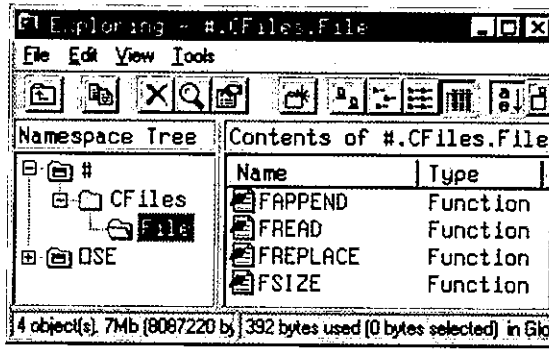
Back numbers of Vector are available from:

British APL Association,
c/o Gill Smith,
Brook House, Gilling East,
YORK YO6 4JJ

Price in UK: £10 per complete volume (4 issues);
£12 (overseas); £16 (airmail) including postage.

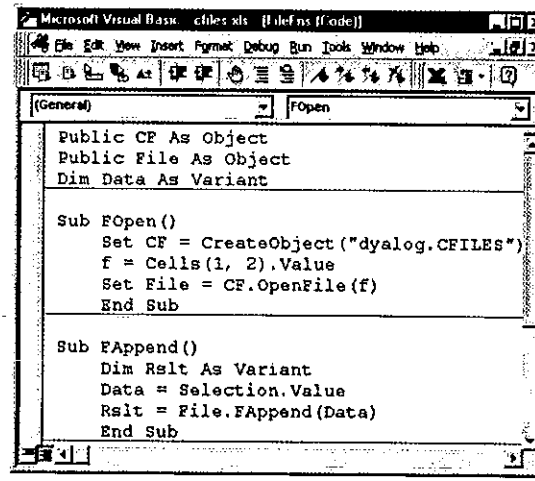
dyalog APL

The Definitive APL for Windows™



Have you ever wanted to call your APL functions from Excel, Visual Basic, or C++ ?

Well now you can!



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

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

MICROSOFT® WINDOWS™ COMPATIBLE

APL99

On Track to the 21st Century

August 10-14, 1999 – Scranton, Pennsylvania

Call for Participation

Papers

Papers are invited on a full range of topics concerning array processing languages (APL, J, K etc.). Papers which emphasize productivity and entrepreneurship are particularly desired. Since this will be the last APL conference before the start of year 2000, when many millennium bugs will manifest, papers which reflect solutions to year 2000 problems are encouraged. Paper topics can include but are not limited to:

- Novel uses of APL in the Business Environment
- APL Legacy Applications
- Can APL be used successfully with Web applications?
- APL readiness for Year 2000
- Distributed APL over a Network
- Remote Procedures and Workspace Concerns
- Mathematical Algorithms
- APL and J in Education

Abstracts for planning purposes are requested by September 1, 1998. Preliminary abstracts are welcome now. Final drafts are due by January 15, 1999. Papers will be impartially reviewed and selected by a distinguished panel of referees. Pending further arrangements, abstracts may be sent to NY/SIGAPL at P.O. Box 2697;New York, NY; USA; 10163-2697, or to David E. Siegel at Siegel@ACM.ORG. The final submission address may well change.

Tutorials

90 minute interactive sessions lead by a PC-equipped instructor. Abstracts due by March 1, 1999.

Workshops

Hands-on sessions in PC-equipped lab facilities. Abstracts due by March 1, 1999.

Vendor Forums

Vendors are encouraged to exhibit their products and discuss new developments.

APL99 is co-sponsored by NY/SIGAPL (a chapter of ACM) and the University of Scranton.

Stephen Mansour — Conference Chair
David Siegel — Conference Proceedings Editor
Garth Foster — Conference Program Chair
Susan Trussler, PhD — University of Scranton Liaison

CORRESPONDENCE

Finding an Elephant

From: George MacLeod 8 February 1998

The son of a friend of mine found this item on the Internet. I therefore have no knowledge of its provenance. I am responsible for the bits on APL and J.

A study was done to find out the different methods computer programmers use to solve problems. The study consisted of taking different programmers to South Africa and telling them to go and find an elephant.

The COBOL programmers immediately adopted a simple strategy; they took a picture of an elephant, faced east and started walking. Each time they encountered an animal they compared it to the picture, and if it did not match they tossed it aside and kept walking. When they came to an ocean they took one step north and started back to the west, repeating the algorithm with each ocean. The more experienced programmers placed a known elephant in Cairo to keep themselves from walking off into the Mediterranean.

The Assembly Language programmers used the same method as their COBOL brethren, but executed it on their knees — insisting it was more efficient that way.

The Database programmers hired thousands of natives at enormous expense, and gave them all a picture of an elephant and sent them running off in different directions. Some of the things they brought back did indeed resemble elephants.

The Artificial Intelligence programmers sat down and developed a complex and complete definition of what an elephant was, and what were its attributes, and the procedure one would follow to create a mathematical ratio of "not an elephant at all" to "absolutely an elephant" for any one animal. At no time did they actually attempt to find an elephant.

The Object Oriented programmers went to the library and took out the book "Finding Kangaroos in the Australian Outback". They carefully changed every occurrence in the book of the words "Australia" to "Africa", and "Kangaroo" to "Elephant".

Because of the power of their language the APL programmers only needed one instruction to find all of the elephants at once. The instruction was: *Africa* ∈ *Elephant*.

Two J programmers, who claim to be reborn APLers, unfortunately took a wrong fork and were never seen again. A planned search had to be aborted as no further J programmers could be found.

37 Newhouse Road
Bovingdon
Herts HP3 0EJ
Tel: 01442 834015; Email: 100412.1305@compuserve.com

APL Null

From: Ajay Askoolum January 1998

APL character null '' (or 0/'*anystring*') and numeric null 0 (or 0/*any numeric vector*) surprised me. I have checked this with APL*PLUS III, APL*Win, Sharp APL and Dyalog/APL: all these interpreters behave identically. Therefore, tempted as I am, I cannot call this a bug. However, I would like to be able to express what is happening in words.

```
''∈'ABCD'      ⍝ Returns null
'ABCD'∈''      ⍝ Returns 0 0 0 0
''⍷'ABCD'      ⍝ Returns 1 1 1 1
'ABCD'⍷''      ⍝ Returns null
```

The results are consistent when 1 2 3 4, say, is substituted for '*ABCD*'.

```
'A'∧.=0/'ABCD' ⍝ Returns a surprise 1
```

as does

```
(0/'ABCD')∧.='A'
```

What is APL doing? The surprise result which prompted me to look into this was

```
2∧.=0/1 1
```

or more generally, 2∧.=*condition*/*value* which caused my function to branch to a routine I had not expected. I would have expected the result to be null or zero.

Can you express what APL is doing in words?

News from Sustaining Members

Dyadic Systems Ltd

Dyadic is pleased to announce the successful launch of a new Dyadic Support Service (DSS) for Dyalog APL. This is primarily an electronic service based on email and the World Wide Web.

One of the key features of the service is that users may download patches containing enhancements and problem corrections from the Dyadic web site http://www.dyadic.com. The process of downloading and applying a patch is completely automated (it's written in Dyalog APL, of course) and convenient to use. Furthermore, the patch files are updated on a daily basis, so DSS users can stay absolutely up-to-date with the latest Dyalog APL developments. In addition, Dyadic's new change control system issues email messages to DSS users, informing them of changes, enhancements and fixes automatically as code is released.

Behind the new service is a totally new *incremental* approach to the development, enhancement and maintenance of Dyalog APL. This technique allows small changes to be made, tested, and safely released to users in a much shorter time-scale than was previously possible. As a consequence, Dyadic is able to deliver a steady stream of enhancements, as well as fixes, that would previously have had to wait for a new software release.

Since the start of the service, some of the enhancements made for the benefit of DSS subscribers are as follows:

Grid Comments	the Grid object has been enhanced to support comments, (tips that appear when the user hovers the mouse over a cell) which are consistent with the latest edition of Microsoft Excel.
:For	the `:For` control structure now accepts multiple variables as in: `:For a b c :in (1 2 3)(4 5 6) ...`
MetaFiles	making complex graphical pictures using the Metafile object is now much faster than before.
Tracer	You can now trace the execution of `⎕LX` on `)LOAD`.
ActiveX	The OCX support has been extended to cater for ActiveX objects that expose an object hierarchy (like OLE Servers).

Syntax colouring Improvements have been made to the colouring of compound names (e.g. *NAMESPACE.SUB.FOO*) and dynamic functions and operators.

Further enhancements in the pipeline include a facility to reformat a function during editing, and a significant improvement in the efficiency of cloning namespaces using □*OR* and of making new instances of OLEServer objects.

Naturally, there will still be new major releases of Dyalog APL issued at roughly 12-monthly intervals, and Dyadic intends to announce some exciting new features at APL98.

Insight Systems / Adaytum Software

SQAPL Client Version 3.0. Available at APL98. Work on the next release of SQAPL (previously announced as version 2.1, now renumbered to 3.0), has now reached a stage where we can commit to a release date: the release will be available by the APL98 conference in Rome (July 1998). We hope to see you all there! Initially, the release will be available for 32-bit Windows (Windows NT, 95 and 98), for the APL systems available in these environments (APL+Win, APL2 and Dyalog APL).

Key features of version 3.0 are:

- No limits on the size of bind variables or record length, except those imposed by the drivers themselves (and "Workspace Full" of course).
- Support for the ODBC ExecDirect, allowing execution of statements without preparing them first, and improved support for stored procedures.
- GetInfo and PutInfo calls, which allow you to inquire about the capabilities of a particular data source, and set options after connection.
- Support for "BrowseConnect", allowing the ODBC driver to provide dialog boxes to guide the user through connection to a data source.
- Bulk Input, allowing insertion or update of multiple rows of data (sets of bind variables) in a single operation.
- For advanced users, we will make the Environment, Connection and Statement handles available so that you can extend the product from APL by making direct ODBC DLL calls.
- Installation Script

Adaytum Software. We believe that Adaytum Planning is en route to becoming one of the biggest APL-based businesses ever. Adaytum Software now employs almost 100 people world wide. In January, our US revenues exceeded UK revenues for the first time, after about eighteen months of operation in the US market, a period in which the UK has also experienced substantial growth. The total number of customers world wide is now in excess of 450.

Our development team has grown to about 15 people, our goal is to more or less double this number during the next twelve months. If you have a knowledge of APL and C++, Visual Basic or Java (or preferably a combination of these), please contact mkrom@adaytum.dk or check our Web site: www.adaytum.dk for an up-to-date list of open positions. We are also recruiting technical writers, product designers and testers. Positions will exist in Denmark, the UK and in North America.

The next major product release is due at the end of 1998. The focus of development for this release is "large, distributed customers": Workflow, Distribution of Reports and Data Entry Tools for organisations with hundreds of widely distributed users. A multi-user Server architecture will provide a Web Server, and an ODBC Server.

Another major theme is "Open Systems": A COM Automation API to allows VB, C++ and other "third party" developers to call our "Planning Server" and build custom planning applications.

Causeway Graphical Systems Ltd

The autumn and early spring have been a period of consolidation and plain hard work for all of us. It may be the approach of 1/1/2000, or maybe the advantages of array programming are again getting noticed. Either way, the result has been that the "Products Department" (Adrian) has been lagging behind the software in getting CPro out of beta and into production.

The good news is that the software itself is proving fast, stable and reliable. Watch the website (the release notes are posted with every significant update) for progress on the tutorial and help files. *NewLeaf* and *RainPro* updates are also available for both Dyalog and +Win to all users on current maintenance — email us for the ZIP passwords, or to receive your copy by diskette if you prefer.

The other good news is that we are now happy to accept both **VISA** and **Mastercard**, which greatly simplifies payment from non-UK customers. Finally, congratulations to Eric Lescasse who has sold more copies of RainPro and NewLeaf than anyone else, including us!

The Vector Product Guide

compiled by Gill Smith

VECTOR's exclusive 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.

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 Systems (Hardware & Software)
- APL and J Interpreters
- APL-based Packages
- 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.

Your listing here is absolutely free, will be updated on request, and is also carried on the Vector web site, with a hotlink to your own site. It is the most complete and most used APL address book in the world. Please help us keep it up to date!

All contributions and updates to the Vector Product Guide should be sent to Gill Smith, at Brook House, Gilling East, York, YO6 4JJ. Tel: 01439-788385, Email: apl385@compuserve.com

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	IBM RS/6000 MD320	11,738	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 APLWIN 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, Manugistics, 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	$150	
	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.
Lescasse Consulting	APL+PC	poa	Lescasse Consulting 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
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	

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
ADAPTA Software	MPS - Master Production Schedulingpos		
	FBS - Forecasting and Budgeting System		
	DRP - Distribution Requirements Planning		
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 & Shareflle	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	*CausewayPro* for Dyalog/W	400/$600	Causeway application development platform for Dyalog APL/W.
	RainPro Business Graphics	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.
	(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.
Lescasse Consulting	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
Lingo Allegro	Internet Server AP	poa	An internet server for Dyalog APL/W. Visit www.lingo.com for a live demo.
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
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
Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
Ajay Askoolum	Consultancy	poa	APL+Win development and migration of actuarial, financial, mathematical applications.
Andrews	Consultancy	poa	APL programming and analysis, Year-2000 legacy systems, algorithms, tree-processing.
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, IS! 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	APL consultancy, modelling and programming.
Lescasse Consulting	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.
Mackay Kinloch Ltd	Consultancy	200-400	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)
Object Oriented Ltd	Consultancy	poa	General APL consulting, code recycling — mainframe to PC, performance tuning.
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.
RadSys Technologies	Consultancy	poa	Areas of expertise: financial systems, risk analysis systems, healthcare systems.
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.
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)	$20
APL Club Austria	Austria	-	Quarterly Meetings	200AS(Indiv), 1000AS(corp)
APL Club Germany	Germany	APL Journal	Semi-annual meetings	DM60
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)
Toronto SIG	Toronto, Canada	Gimme Arrays!	Monthly Meetings, APL skills database, J SIG, Toronto Toolkit	$25

ADDRESSES

ORGANISATION	CONTACT	ADDRESS, TELEPHONE, FAX, EMAIL etc.
ADAPTA Software GmbH	Michael Baas	Marienhoehe 86, 25451 Quickborn, Germany Tel: +49 4106 60977 Fax: +49 4106 67869 Email: 101523.1757@compuserve.com
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
Ajay Askoolum	Ajay Askoolum	42 Hanworth Road, Redhill, Surrey RH1 5HT Tel:01737 771643 Email: 106173.3347@compuserve.com
Andrews	Dr Anne D Wilson	12 Thorny Hills, Kendal, Cumbria LA9 7AL, UK Tel: 01539-731205
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	Curtis Jones (Sec)	228 South 15th Street, San Jose, CA 95112-2150, USA Tel: +1 (408) 292-4060 Email: jonesca@vnet.ibm.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 Email: ssawabini@aol.com, eshaw@aplgroup.com
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	Roolnberg 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) 886-2638; 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 Email: pd@bloomsbury-software.co.uk
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS UK. Tel: 0117-9730036. email: acamacho@clx.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	The Maltings, Castlegate, MALTON, North Yorks YO17 0DP UK Tel: 01653-696760 Fax: 01653-697719 Email: causeway@compuserve.com
Cinerea AB	Rolf Kornemark	Box 61, S-193 00 Sigtuna, Sweden. Tel/Fax: +46 859 255 421 Email rolf@cinerea.se
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	Bartrum House, Ravens Lane, Berkhamsted, Herts, HP24 2DY UK Tel: 01442-878065 Email: gml@simcorp.co.uk
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@ping.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 Ltd	Phil Last	146 Crossbrook Street, Cheshunt, Herts, EN8 8JY UK. Tel: 01992 633807 Fax: 0121 359 0375 Email: phil_last@compuserve.com
Lescasse Consulting	Eric Lescasse	18 rue de la Belle Feuille, 92100 Boulogne, France Tel: +33.1.46.05.10.76 Fax: +33.1.46.04.60.23 Email: eric@lescasse.com
Lingo Allegro USA, Inc	Steven J Halasz	1105 Chicago Avenue, Suite 155, Oak Park, IL 60302, USA. Tel:+1 708 386 8183 Email: sjhalasz@interaccess.com
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: Alastair_Kinloch@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 65 644 Email: oasisbv@pi.net or zwart@oasis.nl
Object Oriented Ltd	Walter G. Fil	Am Grendel 2, CH-6004 Luzern, Switzerland. Tel: 41 41 418 70 70 Fax: 41 41 418 70 77 Email: info@object-oriented.com
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

RadSys Technologies AB	Randolph Schrab	Lovsangarv. 18, S-756 52 Uppsala, Sweden. Tel: +46 18 32 41 53 Fax: +46 708 1996 11 Email:100564.2544@compuserve.com
Renaissance Data Systems	Ed Shaw	PO Box 421, Georgetown, CT 06982, USA. Tel: +1 (203) 270-9729
RE TIme Tracker Oy	Richard Eller	Mikonkatu 8 A, 2.krs, PL 363, 00101 Helsinki, Finland. Tel: +358 9-621 3300 Fax: +358 9-621 3378 Email: re@rett.fi
The Rochester Group Inc.	Robert Pullman	50 S.Union St., Rochester NY 14607-1826, 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.al.x-atom.net
SE APL Users Group	John Manges	413 Comanche Trail, Lawrenceville, GA 30044, 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
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: info@torontoapl.org
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) 872-7806

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
The APL Group Inc	www.aplgroup.com
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
Hoekstra Systems	www.khamsin.demon.co.uk/about/hsl/noframe.html
IBM APL2	www.torolab.ibm.com/ap/apl/apl2.html
Infostroy	www.insight.dk/infostroy
Insight Systems ApS	www.insight.dk
Iverson Software Inc	www.jsoftware.com
Kestrel Consulting	www.kestrel-consulting.com
Lescasse Consulting	www.lescasse.com
Lingo Allegro USA Inc	www.lingo.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.torontoapl.org
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

APL97 CD ROM

reviewed by Adrian Smith

Introduction

The APL97 CD is much more than just a conference programme in a fancy shape — it contains many valuable APL resources, mostly with code that you can easily paste into your interpreter and use in applications. This review is actually more of a catalogue, really just to give you an idea of whether it is worth a few Canadian dollars to get the disk for your own use.

In general, the catalogue on the CD is comprehensive and effective; here is the readme.txt from the root directory:

> *IF YOU HAVE A WORLD WIDE WEB BROWSER:*
> *Select the INDEX.HTM file on the root directory of this CD. You should be able to browse the contents of this CD from here.*
>
> *IF YOU DO NOT HAVE A WORLD WIDE WEB BROWSER:*
> *Get one.*
>
> *PROBLEMS VIEWING FILES:*
> *Check the TOOLS.HTM file for useful pointers. Most of the contributions came in MS Word format, so we have included the latest Word97 Browser for 32 bit Windows, plus a pointer to Microsoft's website for the rest of the possible platforms.*

... actually you can just root around in File Manager and double-click the interesting DOCs. In fact not all the hot-links work, so if the index mentions something then fails to find it, just take an educated guess and go digging. Don't try the MSIE Win3.1 browser installation — instant death and loss of about 15 minutes editing on the text below!

How to Order It

You may order the APL97 CD ROM from www.torontoapl.org for $C20 + shipping, via cheque, money order, Visa, Master, or Amex.

Shipping Costs: please add the following amounts to your order for shipping costs:

Canada/U.S. — $3.00, Other countries — $5.00

You may submit an order from the web site, or for greater security, please FAX your order to: +1-416-781-5732, or send via postal mail to:

The Toronto APL Special Interest Group
P.O. Box 55, Adelaide St. Post Office
Toronto, Ontario, M5C 2H8, Canada

APL97 CD Contents

The following list includes any material associated with workshops and tutorials, and a complete set of abstracts of contributed papers, marked "not on CD" if no extra material is available.

W4: Creating Images in J by Cliff Reiter, Lafayette College: This was a hands-on workshop designed to give participants experience with creating visual images. Exercises for beginners include creation of fractals, plasma clouds and basic image processing. Intermediate and advanced participants are invited to explore their own interests or use available exercises to create 3-dimensional images or symmetric chaotic attractors. The exercises from the session are included as (X: = your CD):

X:\CONFRNCE\REITER\J4IMAGES\J4IMAGES.TXT

T8: An Introduction to J for APLer's by Richard Levine: The subject is the J language and system. The objective is to assist the participants to understand and assess the basic concepts and details. The observed strong commonality of J with many dialects of APL will be exploited to "jump-start" an understanding of J, especially for those who have already made attempts in this direction. We will make heavy use of existing reference materials. There will be ample opportunity for discussion within a carefully planned agenda. Participants unfamiliar with APL will also benefit, and J's unique "non-APL" aspects will not be ignored.

T10: APL+WIN Training Program, OO Programming, etc. by Eric Lescasse, Lescasse Consulting: Published in Vector 14.1 and 14.3. See the workspaces:

X:\CONFRNCE\LESCASSE\QWI.W3
X:\CONFRNCE\LESCASSE\PLAYMIDI\PLAYMIDI.W3

T12: Electronic Commerce, Electronic Data Interchange, and the Internet by Ed Shaw, APL Group: Electronic Commerce encompasses email, FAX, and EDI. Business transactions are conducted over a communications link, whether it be proprietary, a common carrier, a value added network, or the Internet. The tutorial will describe the specific characteristics of each, with an emphasis on EDI, and the role of an EDI translator, such as The APL Group's software product, Qualedi for Windows®. It will also bring you up to date on the involvement of the U. S. Government in EDI. To gain maximum benefit from these opportunities, companies must integrate Electronic Commerce with their transaction oriented computer applications and possibly re-engineer their business operations. This implies an important role for the computer and business consultant.

T13: An APL Toolkit by Richard Levine: The primary subject is the new Release 4 of the Toronto Toolkit, a collection of APL freeware APL utilities now approaching 15 years. The expected audience is those who use APL in a

programming mode, who use or wish to use a "toolkit" approach. Besides a demonstration of new functions (e.g. function development tools) and features, documentation, and architecture of the Toolkit, the tutorial will offer a unique opportunity to present and discuss the software philosophy guiding the Toronto Toolkit. Connections with APL (various dialects) and J language and system features, and other toolkits will also be covered. This tutorial has general appeal to those interested in collections of APL "utility" functions.

T15: Mathematical Roots of J by Roger Hui & Ken Iverson, Iverson Software Inc.: The massive Handbook of Mathematical Functions [1] has now served mathematicians for over thirty years. For several years, work on a companion volume of executable functions has spurred many innovations in the J system. This three-hour workshop provides a review of these developments, presented in a live, interactive manner. Previous knowledge of both the Handbook and J would be helpful, but not essential. Items to be treated include:

Extended Precision Integers. A calculus of Inverses, Derivatives, Integrals, and Duals, for a wide range of standard monadic functions, including families (such as the power functions `^&n`) derived from dyadic functions. Taylor and Weighted Taylor coefficients for a similar range of functions, including rational functions (quotients of polynomials) Smoothing (interpolation) and Quadrature operators for efficient calculation of integrals of arbitrary functions. An operator used (in the form `^!.s`) to provide generalizations of the "rising factorial functions" used in the definition of Hypergeometric functions (provided by the operator `H.`). Elaborate graphic facilities for the presentation of mathematical functions.

T16: How to Write an APL Utility Function by Stephen M Mansoor, the Carlisle Group: In today's business climate, reusable code is essential. But many programmers often don't use existing utility functions because they find them difficult to use or not general enough. Also, they may not know that such functions exist. Instead, programmers often clone lines of code from other functions. This results in sloppy, undocumented code which is full of errors. To avoid this, the author of a utility function must make an extra effort to ensure that his function is designed properly.

T17: GUI Programming in APL+WIN by David Siegel, LEX2000: This talk is intended for APLers new to windows/GUI programming, but familiar with APL. It would cover the event-driven model used by windows, and how to create simple windows applications in APL+WIN, using hands-on examples with several of the APL+WIN GUI controls in place. Assorted APL+Win utilities on:

X:\CONFRNCE\SIEGEL\UTILS97\

(not on CD) P1: SHARP APL Mainframe and the TCP/IP Revolution by Dave Mitchell, Xerox: Sharp APL at Xerox has always been network driven. At Xerox, TCP/IP has brought APL from IPSANET, RJE, TYMNET and SNA into the world of Internet and the Web. Terminal access, file transfer, mail exchange, network printing, Web serving and direct application to application support are some of the successful accesses being done daily. Methodologies, experiences, pitfalls and plans are presented.

(not on CD) P2: Creating Embedded Applications with the NIAL Tools by Mike Jenkins, NIAL Systems: The Nial Tools consist of several compatible versions of the Q'Nial interpreter organized to support the rapid development and deployment of data-intensive applications. The tools are:

Q'Nial, the interactive interpreter for Nial used for program development;
the **Nial Data Engine**, a package for activating Nial applications from other software;
RunNial, a runtime version of the interpreter invoked from the command line;
CGI-Nial, a version of the interpreter to support Web applications.

P3: Data Transfer between Java Applets and Legacy APL Applications by Bruce Amos, Gavin Disney and Duane Sorrey, Reuters: The rise of Internet technologies (particularly Java) provides many benefits for the development and deployment of user interfaces. In many cases, however, the back end system is behind the times: Internet hostile, no object orientation, etc. How can data be transferred between the new generation front end and the old generation back end without compromising the strengths or integrity of either? This paper discusses the use of customised Java data serialisation to achieve this goal against a large IBM MVS Sharp APL system.

P4: APL IDE: A Windows Interface to Mainframe APL Systems by Dennis Paproski, Reuters: When I joined I.P. Sharp in 1988 there were two interfaces to the Sharp APL mainframe:
1. through proprietary terminal emulation software written for DOS called PC108 which emulated an HDS108 terminal as well as providing some extra functionality and
2. through a 3270 terminal or PC 3270 emulation. In 1997 my choices are essentially the same while other PC technology has improved immensely. The goal of the APL IDE project is to provide the Sharp APL programmer at Reuters with a new interface to the mainframe that will improve productivity. This will be achieved through:
1. A Microsoft Windows interface. Multiple function editing windows, drag and drop functionality, search and replace, an improved editor etc. These features are the expected norm for Windows users and are a marked improvement over the current software. The APL IDE should be comparable in ease of use to IDEs available for other programming

languages such as C, Pascal or even Cobol.
2. The IDE will be closely integrated with features available in LOGOS, a software management environment within Sharp APL. This includes an improved interface to a utility library available in LOGOS which will promote the reusability of common code.
3. Improved on-line help for the programmer. Sharp APL manuals will be available through the Windows help facility.
4. Improved debugging tools, a visual step debugger and watch windows.
5. Access to powerful APL tools on the mainframe through a GUI, ie. WSDOC, or quad FM. The APL IDE will be mostly written in Dyalog APL for Windows with some functions written in Microsoft Visual C++ available through a DLL. The development of the IDE will also provide additional benefits beyond improving programmer productivity: 1) We can add additional functionality through further upgrades i.e. sockets connections. 2) We can expand the usability to other platforms i.e. Windows NT or Unix.

P5: Computer Construction of Weaving Designs by Keith Smillie, Univ. of Alberta: J algorithms are developed for deriving the weave of a piece of cloth from the instructions for setting up a loom, for the converse operation of deriving the setup instructions from the weave, and for introducing colour into the weave. Published in Vector 14.3.

P6: About Recurrent Calculations in APL by Andrei Buzin, RusAPL: It is known that APL gives the possibility to think globally, in terms of arrays. It has powerful operators which process arrays as a whole. Unfortunately, not all problems of array processing can be solved elegantly by APL. APL is oriented to regular array processing, which means that all elements are processed similarly. However from time to time the researcher who uses arrays requires singular or recurrent array processing, in which the algorithm of single element processing depends either on this element itself or on the other elements of array. As examples of singular processing we can mention calculating the logarithm of a numerical vector or the derivation of a piecewise-smooth function in points among which there are the points of non-differentiability. It is obvious that the points where the function or operator is not defined must be excluded from the processing. APL does not do this, nor inform us which element can't be processed. To obtain such a result we must use looping, but loops in any interpreter work very slowly. It would be fine if this iteration was programmed as part of the language, as in the case of reduction, scan or each operator. In this article we shall discuss the other type of irregular array processing, namely, the recurrent calculating of the elements of array. See:

X:\CONFRNCE\BUZIN\RECURE\RECURE.DWS

P7: TimeSquare Tables — A New Data Type by Doug Forkes, Soliton Associates: While developing the TimeSquare project, we found it quite useful to define a special datatype, and a set of functions to act upon it. The new datatype is somewhat analogous to an SQL table, so we call it a table. A table can be thought of as essentially two-dimensional, with a finite number of ordered rows, and an infinite number of identified but unordered columns. The identifier of a column of a table may be any APL object. Each element of a table may be any APL object, or may be null. This paper will describe a set of APL functions to manipulate such tables, and provide some examples.

(not on CD) P8: Global Limits Control System at Deutsche Bank by Michael Kornacker, Deutsche Bank AG: GLCS (global limits control system) is a part of risk management in the business with financial institutions counterparty and country risk monitoring in the trading products mm, fra, fx, spot, fx forward and commercial paper; history, present, future.

P9: Interactive Design of Structures — A Program for Everyone by J. Riebenbauer & J. Hoffman: We will present Intraplan V2 — a graphical DyalogAPL/W application for the design and analysis of planar member frame structures. We will demonstrate how to input and optimize a load bearing structure using the complete visual and intuitive GUI of Intraplan. We will also talk about the program development history, e.g. about our experiences we made during the migration from APL*PLUS II to DyalogAPL/W, with its object oriented Windows interface and namespaces. Published in Vector 14.2.

P10: Alvin Surkan Univ. of Nebraska-Lincoln: A Concise APL Function View of a Constructive Algorithm for Neural Networks that Generalize: APL functions are provided as descriptions of constructive algorithms for optimizing the synthesis of neural networks while improving their generalization capabilities. Programs of these algorithms construct networks of binary weights for classifying or partitioning sets of arbitrarily-high dimensional binary patterns in the closer of two classes. The study of constructive algorithms for identifying such networks is of direct interest to designers who build array processing hardware classifiers from fast two-level digital circuits. Provided is an APL exposition of a constructive algorithm for synthesizing minimal neural networks. One of our objectives is to introduce creative APL users to this emerging application area and the language's potential for describing array-based software and hardware. This constructive method incorporates a minimum overlap pattern separation and a target switching algorithm. Prototypes of constructive algorithms implemented with typical , scalar-based procedural languages typically require hundreds of statements. Array based formulations with functional style programming languages like APL and J require a few short functions. See ...

X:CONFRNCE\SURKAN\ASURKAN.DWS

(not on CD) P11: J vs. Mathematica by Murray Eisenberg, Univ. of Massachusetts-Amherst: APL and J have had limited success in penetrating mathematics education, especially at the college level. By contrast, in recent years Mathematica has received increasingly wide acceptance. Why? How do J (as a representative of the APL family of dialects) and Mathematica stack up against one another — as tools for getting answers; as programming languages; and as tools of thought? These are among the questions to be addressed by the presenter, who has taught linear algebra for many years using APL or J and, most recently, using Mathematica.

(not on CD) P12: Nested Array Internals and Efficiency by Roy Sykes, Sykes Systems: The advent of nested arrays has given APL users an even richer repertoire of ways to store and manipulate data, especially for small or ad hoc problems. But heavy usage of large datasets still demands efficient processing. By examining the internal architecture and implementation of APL, we can evaluate different storage strategies to predict their efficiency. This talk will uncover these hidden aspects of APL and use several examples to illuminate our findings.

P13: APL and Nested Arrays — A Dream for Statistical Computation! by Alan Sykes & T. Stroud, Univ. of Wales, Queen's Univ.: Many papers have been produced in the last 10 or more years extolling the virtues of APL for statistical computing. Such papers have stressed APL's array-handling operations, the use of user-defined operators, and the often transparent flow from mathematical notation to APL's equivalent computational notation. The use of nested arrays figures relatively little in this. (Of course, the current Windows-oriented computing medium means that a function's argument may well be, often is, a long nested array!) This paper demonstrates why APL with nested numeric arrays is just what the statistician needs in order to deal with missing data in a sample survey analysis. One technique of dealing with missing values is that of multiple imputation in which each missing value is replaced by a set of between five and ten 'typical values'. APL with nested arrays provides just the right medium for the efficient storage of the resultant database. Moreover, because array operations on nested arrays employ scalar extension, the paper shows that typical statistical functions, such as the calculation of means, standard deviations, regression estimates, can be programmed, often with little or no change, so that they will operate on the nested database to provide multiple answers (each answer corresponding to one of the chosen typical values representing the missing values). The multiple answers can then be used to give answers required that truly reflect the variability induced by the missing data. The statistical knowledge required to appreciate the content of the paper is minimal — the beauty of APL in the service of Statistics is considerable! Published in Vector 14.2.

(not on CD) P15: Evolution of CoSy by Bob Armstrong, Coherent Systems: CoSy is the NoteComputing environment I have evolved since meeting APL in the mid '70s, for general support in the business of life. Currently CoSy is constructed in old STSC flat PC-APL in 1984 as an open-source hypertext browser of its own objects — although that vocabulary was not yet in use. The target now is to rip the organs of Window95 (OLEs) open into a coherent linguistic structure down to the hardware — creating an environment providing the user/programmer complete control over their notebook computer. The practical next step is to integrate CoSy with a competitive APL language community and all the algorithms its members are interested in sharing.

P16: War on the Workspace — a Databased Commercial Application by Ed Shaw, APL Group: How do you properly support thousands of dispersed users of a PC application without losing control and yet provide timely and apparently customized features? How do you continue to maintain and enhance a commercial application over a period of many years, including a conversion from DOS to Windows and the inevitable changes in personnel without falling off a cliff? There are probably as many answers as there are developers to this question. The APL Group has chosen to ignore the workspace and to store all control data functions in a highly structured manner in its proprietary database written in APL. It has also adopted a number of procedures and controls which have allowed it to continue to succeed over 14 years with a continually evolving product. The techniques adopted will be described with the hope that some of them may be of use to others and also that they may stimulate a discussion of alternatives for the mutual benefit of all who attend.

P17: J Phrases for Statistics at a Glance by Gilliro Suzuki, Institute of Statistical Mathematics: In my talk, many functions for statistical data analysis are presented. These functions are mainly written by tacit definitions. These are then convenient as a J primer. Main contents are as follows:

mean, median, mid-range, moving average, range
mean deviation, standard deviation, coefficient of variation
mean difference, quantile deviation
dispersion measure for categorical data
histogram, frequency polygon, stem-and-leaf diagram
correlation ratio, Pareto curve
scatter diagram, correlation coefficient
regression analysis, principal component
analysis of variance, discriminant analysis, etc.

P18: J Installed String Manipulations Applied to English-Esperanto Machine Translation, by Toshio Nishikawa, Chiba Institute of Technology (Toshio Nishikawa nisikawa@ism.ac.jp): J has been widely used for

mathematical computations. However, it's less often used to do string manipulations such as machine translation. The author developed a small machine translation system using APL, and tested its adaptability through English-Esperanto and Japanese-Chinese translations. Given the growing popularity of J, we made a plan to convert the system to J. First, we thought it could be easily converted in line-by-line code, but we soon realized this to be impossible. Therefore, we rebuilt it as almost a new version, starting from making several function tools for manipulating character strings, although we aim for machine translation. We adopt English-Esperanto as a pair of translation, because of simplicity and regularity. Thanks to the power of J, especially 'box' and 'each' codings, we can expect a compact translation system.

D1: Chaos with Symmetry — a visual exhibit by Nathan Carter, Richard Eagles, Stephen Grimes, Andrew Hahn, and Cliff Reiter, Lafayette College: In recent years mathematicians have found ways to construct images of attractors resulting from function iteration, that have both symmetry and chaos. We extend, explore and refine those methods to allow us to look at attractors with rotational symmetries, shift symmetries and tiling symmetries. The images in this visual exhibit were constructed in J and selected for their provoking appearance. See ...

X:\CONFRNCE\REITER\CHAOSSYM\FIG?.GIF

APL97 Software Library — contributions of software or documentation

Contributor	Contents
Mike Jenkins	a file-based version of the Q'Nial Web site
Mike Jenkins	a demo version of Q'Nial for Windows, complete with libraries
Richard Levine	programs for transferring functions and data between various APL systems (some restrictions apply)
Keith Smillie	"Beginning J" — an article in PC Word 6 format, with two script files
Jim Weigang	APL Newsreader — the complete archive of comp.lang.apl articles from Feb 89 to the present, along with Jim Weigang's APL Newsreader for browsing and searching the articles. (Run-time APL interpreter included.)
Jim Weigang	APLASCII Workspaces — utilities for converting APL symbols to {keywords} and back. Great for imbedding APL programs in e-mail and newsgroup postings and for moving workspaces between different APL systems. Versions are provided for all major contemporary APLs.
Jim Weigang	Jim Weigang's Web Site — described by Vector as the "definitive APL home Web Site pages", this website has nearly a megabyte of useful information, example programs, and amusing diversions. Browse it from the CD without the usual Internet delays.

The Waterloo Archives

The complete contents of the Waterloo APL/J Archives, as of July, 1997

APL Shareware Review: ComLog Comic Logger

reviewed by Jon Sandles

Sometimes I have cast my eye down the Vector product list and seen things that sound quite interesting, but I have no idea what they are. ComLog's Comic Logger is listed in the OTHER PRODUCTS section and is described as "APL*PLUS II comic-book inventory system".

Intriguing stuff, I am sure you will agree, so I thought it would be nice to obtain a review copy of this software just to see what it was. I e-mailed the creator explaining I wanted to review ComLog for *Vector* and shortly after I received a review copy over e-mail. Everything installed just great, even under NT4.0, although I did experience some random hangs with the software, but I put this down to NT's DOS session being very much not a real DOS box.

The software is a DOS menu-driven system, in fact, rather good for a shareware DOS product. It is basically a database product for cataloging Comics. What is so special about comics you might ask - why not use a generic database product? Comics have some interesting relations in their database structure, things like the fact that you get the same artists (wrong technical term probably but I do not want to get into that!) working on the same story lines most of the time, but every now and again switching to different story lines, the same artists and writers also tend to stay together in the same "teams" but again not always. The whole area of re-issues of past stories in different comics also causes great problems, so cross referencing to old issues needs to be possible. APL was presumably the perfect tool to provide the search facilities that allow the user to find given artists, price ranges and story lines.

The product is presumably invaluable to both professional and amateur comic collectors and although it is DOS-based it is cheap enough to be a good starting point in setting up a comic database. You would be hard pushed to produce an application that has this sort of functionality using just Microsoft Access for example. And, of course, if it helps publicize esoteric transatlantic comics like the mighty Silver Surfer it has to be a good thing!

If anybody else fancies reviewing a piece of APL software mentioned in the product guide, we would be happy to obtain review copies in exchange for a couple of paragraphs.

Dyalog APL for Motif Version 8.1 Release 2 for Sun Solaris 2.5

reviewed by Bob Hoekstra (bob.hoekstra@khamsin.demon.co.uk)

Abstract

Dyadic Systems had been a major force in the UNIX APL market for some years when they left it to concentrate on developing APL interpreters for Microsoft Windows. With their latest offering they again target the UNIX users, giving them the benefits of the last few years of Windows development.

Introduction

Those readers who have met me (even only briefly) will know that I am very enthusiastic about two things in computing: APL and UNIX. When these two interests combine (as in this case) I can become positively fanatical.

At home I run Solaris 2.5 on a Sun SPARCstation 2 (the SPARC processor upgraded to a Weitek Power µP). This machine provided the test bed for the software under review, *Dyalog APL/M 8.1.2 for Sun Solaris 2.5*. I also have PCs with Linux (the Debian distribution), Windows NT 4 and Windows 3.11. This is relevant as they provide a basis for comparison — Dyalog APL/M is a UNIX port of APL/W 8.1, which I use under Windows NT.

In fact, I use several versions of APL on many platforms (including mainframes when my clients require this). I have been waiting to get my teeth into this particular APL interpreter ever since I heard of the APL/M project.

This Review

This is an attempt at providing a comment on a new product. It is targeted at an audience of APL users, many of whom will probably know little or nothing about UNIX. I will try to provide adequate notes and explanations where necessary.

As a separate exercise, I will also be publishing a very similar review in *Browser*, the magazine of the Sun User Forum (previously known as the Sun UK User Group). This will cater for the UNIX (mainly SunOS) users who may be unfamiliar with APL. Of course, there may be a small number of readers who,

like me, use both APL and UNIX. Some time after both publications appear in print a version of this review will also appear on my web site, and this version will be kept as up-to-date as my circumstances (and Dyadic Systems) allow. See www.khamsin.demon.co.uk.

Readers are assumed to be familiar enough with Microsoft products not to require any explanatory notes.

Please note that the review is not intended as a vehicle for expressing my views on the qualities of the interpreter itself (which are very favourable) but rather the implementation of that interpreter under the Solaris 2.5 operating system.

What is UNIX?

Note: This is a quick introduction to UNIX. Those readers familiar with UNIX or UNIX-like operating systems may want to skip this section. Please feel free to do so.

UNIX is an operating system with a heritage dating back to the 1960s. It originated in the AT&T laboratories in the USA, but it was a long time before it was offered as a commercial product, first having been taken on board by many universities, scientific laboratories, etc. Each institution added its own features, and as a result the standard UNIX distribution has become an interesting combination of smallish programs, which are not all consistent in their interface. The general idea in UNIX is to have a smallish kernel which provides essential services and to have separate simple programs or tools to provide everything else. This is in sharp contrast to DOS and related operating systems, where all commonly used functions reside in the kernel.

There was some competition as to what a standard UNIX distribution should consist of and this eventually concentrated about two rival camps: those who followed the AT&T line of thought and supported what became known as *System V* UNIXes, and those who followed the adjustments favoured by the University of California at Berkeley and supported *BSD* (Berkeley Software Distribution) UNIXes. In many cases, one would use the same or similar command to perform a task on the two types of system, but the command would react slightly differently. Sun Microsystems' operating systems up to and including SunOS 4 were BSD compliant. However, SunOS 5 (the operating system that underpins the Solaris 2 product) is a System V Release 4 (SVR4) compliant UNIX. SVR4 is becoming the "standard UNIX", with other vendors following suit.

At first there was no UNIX GUI interface, but soon X Windows (developed at MIT and descended from work done at the Xerox Palo Alto Research Centre or PARC) was adopted almost universally. Unlike the Microsoft operating systems, no UNIX vendor chose to "integrate" the windowing system with the operating system itself. It remains a separate entity, necessary if support for terminals and terminal emulators is to continue. Initially there was much diversity between different vendors' implementations of X Windows (see Notes 4 and 6) but recently many vendors are becoming CDE compliant (see Note 9).

Why use UNIX?

I could write a separate article on this subject, but I think I'd better stop at a short list of some of the reasons why (in my opinion) UNIX is a better choice than any Microsoft operating system for the APL developer. Note that none of these features are new but have been in UNIX for years. Some of them are slowly creeping into Windows NT, but (my opinion again) the implementation under NT is *never* as elegant or easy to use.

1. UNIX has always been a multi-user operating system. This means that more than one user can use the computer simultaneously. Proper account is taken of the rights of different users and their ability to protect their data from, or share it with, their fellow users. Only the superuser, root, would normally have the ability to override this security system.

2. The file system supports decent locking facilities which may be used in conjunction with the locking capabilities of the APL interpreter, simplifying the writing of multi-user applications.

3. UNIX is network aware. Proper security exists for logins and resource sharing across networks (this includes the Internet) and so networked applications can be written simply.

4. A version of UNIX is available for almost every computer imaginable (please don't bombard me with exceptions). Some examples:

    a. SunOS 5 (Solaris 2) runs on PCs ('486 or better) right up to 64-processor SPARC machines. Sun have announced their intention to provide a far greater number of processors (around 1,000!) soon (new hardware, same operating system). And Microsoft say that NT is scalable?
    b. Linux (a free UNIX-like operating system) will run happily on a 4MB 386 machine, SPARC boxes and Alphas. A Macintosh port is in progress.
    c. Talking about Apple, their latest operating system, Rhapsody, is a derivative of what used to be NeXTStep, a Mach-kernel UNIX.
    d. Even IBM have a UNIX (AIX) and I once saw a report of an OSF 1 derivative UNIX that IBM had running on a 360 mainframe!

5. UNIX systems have a single file system. Devices (hard disks, CD ROMs and perhaps even floppies) are "mounted" at points in the file system so that the file system becomes a cohesive single unit. This is extended to remote disks, where machine A may allow machine B to mount part of its file system, which then becomes part of B's file system also. Note that this is (to the user) a much simpler concept than "mapping" a remote machine's shared directory to a logical device name as is done under Windows or NT, and that the security implications have (in my humble opinion) been resolved in a far more flexible and satisfactory way. Along with the ability to create symbolic links (see Note 8) and automounting (see Note 13) this makes for a very powerful file system.

6. While there are significant differences between one vendor's UNIX and another's, they also share many similarities so that they will very happily share resources over a network. I have worked with networks where machines running SunOS (4.x and 5.x), HP-UX (Hewlett Packard's UNIX), IRIX (Silicon Graphics's UNIX), and Linux all share resources in such a way that the users were (blissfully) unaware of the physical location of the disk they were reading from or writing to.

7. The window system (X Windows) is also network aware, allowing the running of a program on one machine with the user interface displaying at another machine altogether. This allows the developer to make full use of the client-server concept. Note that the "display" machine need not be a UNIX box — a PC running Windows 3 and an X server will do nicely.

8. Administration is a lot easier on a UNIX machine. I'm not talking here about opening windows and clicking on buttons to create a new user, but the resolution of real problems which occur in real life. In particular, the unknowledgeable user can be prevented from rendering his machine unusable (e.g. like deleting parts of the operating system) while retaining the power to personalise his own working environment in a safe way.

9. UNIX is a mature, stable operating system. From a recent survey, it was noted that no users of Solaris 2 on PCs had ever seen the equivalent of NT's "blue screen of death". Not one. Yes, crashes occur, the user may get locked out or applications die in disgrace, but no-one had seen the operating system roll over and die to the point where the only solution was a physical power cycle (which is potentially fatal to a cacheing file system like that of NT or almost all UNIXes). The worst I have ever seen was where I (as system administrator) had to login remotely and force a user off the system, which closed all his open files and the system became usable again (I did reboot the system after this, but this was probably not necessary).

I have really only scratched the surface here. If pressed, I may write a further article justifying my comments, but that would be of little interest to APLers in general.

A little fact that may be of interest is the early tie between UNIX and APL. Apparently Ken Thompson (one of the great names of both UNIX and C) wrote an APL interpreter, APL/11, while at Bell Labs. I have no idea when this was, but to quote Michael Cain (who currently maintains APL/11) "...it spent some time at Yale and finally arrived at Purdue University. Since 1976 it has been modified by Jim Besemer and John Bruner at the School of Electrical Engineering, Purdue, under the direction of Dr. Anthony P. Reeves...". This interpreter lives on as a free APL for UNIX and Linux, and in its latest incarnation as FreeAPL, for Windows as well. Any who are interested may contact the "Licensor" of FreeAPL, Tauno Ylinen, Helsinki, Finland (email: tylinen@mits.mdata.fi).

About Dyalog APL

I first came into contact with Dyalog APL in 1990, running version 5 under AIX on an IBM 6150 RT/PC (see Note 1) using dumb terminals. I immediately realised (despite limitations in the hardware) that this was an excellent implementation of the language, in many ways superior to the VSAPL, APL2 (see Note 2) and APL*Plus (see Note 3) that I had been using in the past. Shortly thereafter I enjoyed using Dyalog APL/X version 6 under SunOS 4 with OpenWindows (see Note 4).

I have subsequently used the Dyadic products under several UNIX variants (see Note 5) as well as MS-DOS and Windows (3, 95 and NT) and have always been impressed by the interpreters. Prior to Dyalog APL version 7, Dyadic Systems' products had been mainly for various UNIXes. They switched their main development effort on the Windows platform after Version 6. Version 6.x was ported to DOS and later to Windows, and this developed into version 7. Version 8 came later and is intended for 32-bit Windows (i.e. Windows 95 and NT). As far as I am aware, there was never a Dyalog APL version 7 for any UNIX operating system. Perhaps this was not surprising as version 7 saw the introduction of a GUI programming interface under MS Windows. Nevertheless, there have been interesting non-GUI-related developments in version 7 and version 8 . When I heard that version 8 was to be ported back to the UNIX operating systems, I breathed a sigh of relief.

Currently available are version 7.3 for 16-bit Windows (i.e. Windows 3) and version 8.1 for 32-bit Windows: Windows 95, Windows NT and now some versions of UNIX using X Windows, with restrictions on the window managers (see Note 6) in use.

Dyalog APL 6.2 (the character version) is also still available for many UNIX platforms, though APL/X 6.2 (the X Windows version) is no longer sold — its

place in the product line has been taken over by APL/M. I suspect that a non-X version 8 will appear soon to offer the non-GUI related benefits to those developers and users who do not need the Gui.

The Software

The interpreter supplied here is, as far as I can tell, identical to the version 8.1 supplied to Windows 95 or NT users. It is an ISO 8485 compliant APL interpreter and development environment with the usual Dyalog extensions to the ISO standard. The latest of these extensions include namespaces, dynamic functions, coloured function and variable editors and a complete implementation of the MS-Windows GUI tools as implemented by Dyadic Systems in the Windows 95/NT product.

Some readers may be surprised by the last addition. The Windows GUI tools are implemented by the use of the MAINSoft Corporation's MAINWin Cross Development Kit (CDK). This CDK is based upon Microsoft source code ported to UNIX and is constantly updated as new features are added to Windows. Dyadic Systems stress that MAINWin is *not* an emulator. It executes native machine language and makes direct calls to Xlib.

Use of this CDK entails paying Microsoft a royalty for the use of their code. This is part of the purchase price and is done by the suppliers, not the purchaser. This is no doubt the reason why a freely distributable run-time interpreter, which cannot be used for development, is *not* included in the package. In Windows versions this run-time interpreter is a big selling point.

The Package

The review copy of Dyalog APL/M arrived on a single compact disk, accompanied by some 10 A4 pages of installation and setup instructions specific to the platform. A purchaser would also receive the same documentation that accompanies an interpreter intended for Windows, which consists of 3 soft-cover books of good quality as well as some additional material. While this is very good documentation for those familiar with APL, it is probably inadequate for those wanting to learn the language, who will have to turn to more suitable literature or attend a language training course.

Loading the Software

The instructions are clear and easy to follow. The installer should ideally have root access. A quick `cpio` command and a little less than 35 MB is installed on

the disk (in my case, in `/usr/mdyalog`, but see Note 8). This took just a few minutes on my machine, even though I was NFS-mounting (see Notes 13 & 14) the CD from my Linux box over my home network.

Next you are instructed to edit the `apl.ini` file in order to set the values for LPTn so printing can be performed. While this may seem disappointingly "Windowsish" to the readers, let us remember that APL/M is a port of the APL/W product. Only PostScript printers are supported. Lastly, a shell script (see Note 7) has to be executed for each user in order to set up their environment, and you are ready to run.

I would have been a little happier if the disk had contained a Solaris-style package, which I think is an extremely neat way to install and remove software. However, given that this software is available for several UNIXes, most of which do not support packages, I am quite happy that Dyadic Systems made the installation as simple as possible.

Running the Software

I symbolic linked (see Note 8) `/usr/mdyalog/mapl` (a Bourne shell script) to `/bin/mapl` so that I didn't have to adjust my `PATH`, and entered this command in a `dtterm` session (I usually run CDE). The software immediately started creating a font cache, analysing each of 1254 fonts installed on my system when running CDE! (see Note 9). This took several minutes, but fortunately this is not repeated on subsequent loads, but it may be repeated if the fontpath changes. This happens to me if I change to the `twm`, `olwm` or `olvwm` window managers.

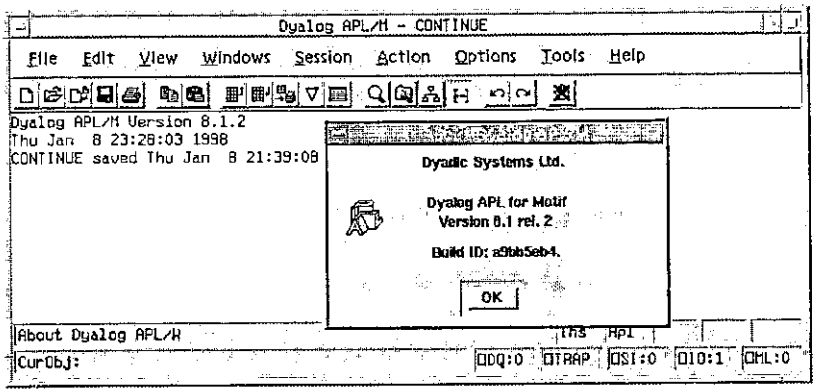


Figure 1: The Session Manager and the "Help/About" box

After a short delay, the APL session appears, complete with motif adornments (see above). The window adornments are hard-wired into the system, although you do have the option of choosing a "Windows Look" (see below). Even if using OpenWindows (`olwm`, `olvwm` or `twm`), the user sees only Motif or Windows widgets attached to the APL session manager, and trying something more exotic by way of window managers (see next section) really brings problems.

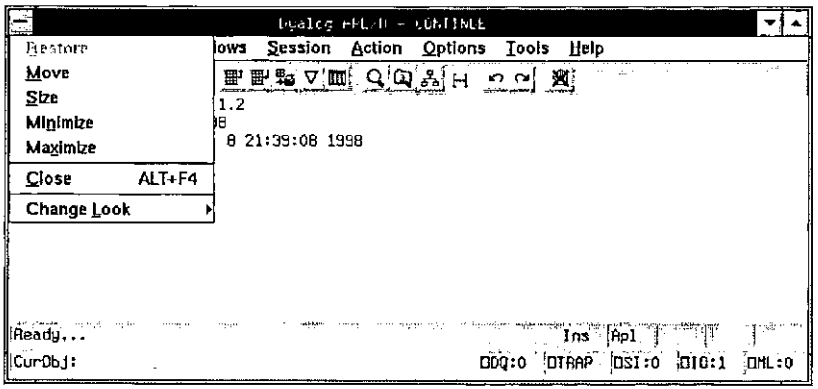


Figure 2: The "Windows Look" — about to return to the "Motif Look"

It surprised me that the "Windows Look" was in fact that of Windows 3 (or NT 3), and not Windows 95 (or NT 4) as Windows widgets are included in the software which are available in APL/W 8 (for Win95 and NT) and not in APL/W 7 (for Win3). I asked Dyadic systems about this, and they explained that future versions of MAINWin will provide the more modern "look". This is part of the continuing development of the product.

Trying Other Window Managers

Dyadic Systems warns that *only* the following are supported:

Supported X servers

a. X11 Release 4 or 5 on any supported platform
b. Xnews on Sun with OpenWindows 3
c. Hummingbird PC X WServer eXceed version 5.1.3.0 or greater

Supported window managers

a. mwm version 1.2
b. olwm on Sun
c. vuewm on HP HP-UX
d. 4Dwm on SGI IRIX
e. CDE 1.0

From my experience, `olvwm` (on Sun) also works and I experienced no problem with `twm`, although Dyadic Systems tell me that this last window manager has caused problems.

I tried to run the software on my Sun box, displaying on my Linux machine (using XFree86 and several window managers, including `afterstep`, `fvwm2`, `kwm`, `olwm` and `wmaker`), but failed. The mouse pointer turned into a black rectangle and the colours were all wrong. In particular, white was sometimes turned into black, but not the reverse, meaning that I was often left with black text on a black background! I am told that XFree86 does not implement the complete set of X intrinsics.

I would have liked to try an X terminal as well as a PC X Server, but I couldn't due to lack of availability. I also cannot confirm whether I would have been able to display on the other supported configurations, e.g. running on a Sun and displaying on a Silicon Graphics or Hewlett Packard box because I did not have access to the hardware required.

Performance

I had been warned that the hardware I was using was not really powerful enough for displaying APL/M at its best. Nevertheless, as I had run other versions of Dyalog APL on slower machines, I was keen to try it. I had a copy of APL/X version 6.2.0 at my disposal, and this seemed like the ideal benchmark for the new product.

It was obvious that APL/M takes a long time to start. Repeatedly timing the start gave about 35 seconds for the first time after logon, and about 20 seconds thereafter. I am sure a more powerful machine would have helped here! However, this is something a user does only once or twice per day, so is probably not very important. It certainly becomes insignificant compared to the time taken to start CDE or OpenWindows. By comparison, APL/X was up and running within 2 seconds on my SPARCstation 2.

Once up and running, differences between the two version on my SPARCstation evened out. My tests were not exhaustive, but it appears that APL/X is about

50% faster on simple arithmetic operations and almost twice as fast on creating some (not all) X windows. Again, I mentioned this to Dyadic Systems, who tell me that they are aware of the problem and intend to resolve it. They are confident that APL/M will soon be as fast if not faster than APL/X.

Bearing in mind that the new interpreter (version 8.1.2) has a lot more functionality than the older one (version 6.2.0) and that the original version 8 had been written for a totally different platform, I had been expecting some differences. However, these differences were much larger than I had anticipated.

Someone expecting to do development on this interpreter or use it on large amounts of data is likely to have a more modern workstation and the speed differences would be less important. Generally, I found it quite usable on my machine, except for the initial start-up time, and I don't see execution speed as a serious problem, although it might be in certain circumstances.

Porting Code

In the past, porting Dyalog APL code from one platform to another had been a problem. Workspaces had an incompatibility over little and big endian machine architectures and had to be ported between workstations (see Note 10) and PCs. This version of the interpreter can read either little or big endian workspaces, transparently converting the workspace on reading from disk. This means that I could read workspaces saved with APL/X without a problem. However, porting the opposite way (APL/M to APL/X) is more problematic, as they are not compatible. However, I doubt that anyone would want to do this (well, not often, anyway).

Even more useful is the fact that I could swap workspaces with my APL/W on NT without any problem at all in either direction.

Unfortunately, component files (see note 11) are not portable between architectures. To me this seems more important than providing workspace portability, as this means that sites using both architectures will have to maintain separate data sets for each! Utility workspaces are provided which make the porting of component files simple. There are alternatives to this (i.e. creating an APL data server — TCP/IP code is also built into the interpreter and very easy to use). I asked about this and was told that Dyadic Systems intend to implement a platform-independent component file system for both the Windows and the UNIX product in a future release.

I tried running several workspaces that had been created with APL/X and had no problem with any except those that tried to implement X11 graphics routines.

This functionality is not included with APL/M. Again, this is not surprising given its heritage, and other ways of producing similar graphics are provided (see Figure 3). *Rain* graphics (from Causeway) also worked without problems.

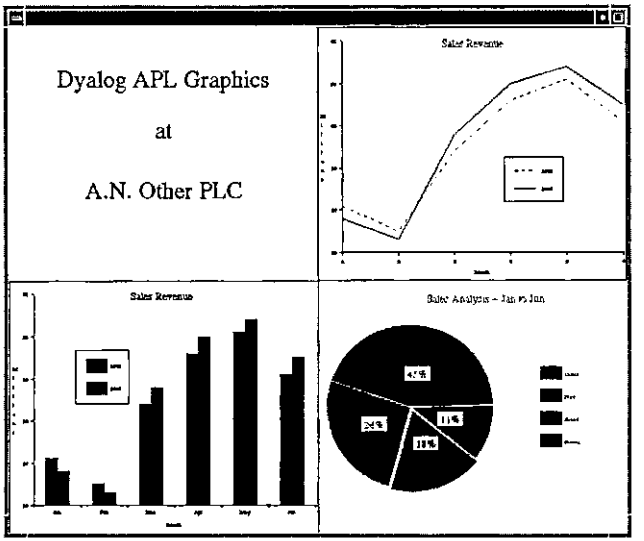


Figure 3: An example of the graphics available

Next I tried running code produced using APL/W under NT. This produced few problems and was, on the whole, surprisingly simple. As a final test, I ported a large application (the result of about 3 months' work) from NT to Solaris. This took me about 24 hours, or 3 man-days. The result worked, although rather slower than on my NT box. It is difficult to be objective about the performance as my NT machine is a modern, fast PC (Cyrix 686 P150+ with 96MB RAM). I suspect that, had I been using an UltraSPARC machine, the performance would have been quite similar.

The following are some of the danger areas I ran into:

Path separators

APL/M has to have UNIX (/) rather than the DOS (\) path separators. Readers may think this obvious, but Dyalog APLs for DOS and Windows understand both! This meant that, where you were asking the interpreter to parse a text string that was obviously a path, you could use either forward or backward slashes.

Of course, code that was meant to run on either platform should be written in a platform-independent way, but if you are porting code that isn't, it could take

some time to correct. Note that both slashes have specific meaning in APL code, so a simple "search and replace" will not do.

Executing operating system commands

Again, this hazard may seem obvious. In a hurry, I had produced sloppy code which obtained the name of my current directory by capturing the response from the DOS `cd` command. When the same workspace did not run successfully under UNIX, it took just a few moments to find the problem: `cd` is also a UNIX command, but the effect on my APL code was quite different!

Reading ini files

One has to be careful with taking these files from a Microsoft environment, as they contain carriage return/line-feed pairs terminating each line. In a UNIX file system, the carriage return must be removed (or the code which reads the file must remove it) since the extra character can cause problems. On the other hand, I was pleased to find that code using Microsoft utilities to read and write these files worked without any problems.

Case in file names

This is a tricky one, as many of the DOS utilities that return a file name may do so in upper case. Of course, DOS/Win is not case sensitive, so it doesn't matter there, but UNIX is. Furthermore, if I move a file from my PC to my Sun box using `ftp` it comes across in upper case, whereas if I boot the PC in Linux and NFS-mount (see Notes 13 & 14) the NT partition is in lowercase! The whole problem is not as trivial as it seems!

Conclusions

Dyalog APL version 8.1 provides an excellent implementation of the APL language, and this port maintains that standard. Consequently, this is a great choice for sites wanting to port existing APL (especially the most recent version of Dyalog APL) from Microsoft operating systems to any of the supported UNIX systems.

For APL development on a UNIX platform, APL/M provides an easy-to-use interface which lends itself to high productivity from the developer. The creation of GUI interfaces is particularly easy, and the code will be easy to port to a PC should this ever be necessary. Note that the PC version includes a freely distributable run-time interpreter which is not provided with the version being reviewed. Developers would benefit from fairly powerful workstations.

For sites which have high power requirements, processing a lot of data, possibly with a multi-user system running on large UNIX servers, the older APL version 6 may prove more successful as it is faster at present. However, this would mean missing out on some of the excellent recent developments in the language. Personally I am confident that the performance issue will be solved soon.

Lastly, I was expecting just a little bit more from this product. The fact that I'm limited in my choice of window manager is irritating, as is the failure to let me display on my Linux machine's X server. I plan to take up this matter with XFree86, the producers of the X server used under Linux, and this may still be resolved in a future release. I like many of the features, and even admire some that I don't like very much. Fancy making an X application look like an MS-Windows one on my X screen!

On balance, I feel that Dyalog APL/M is a success. I am confident that it will become even better as it matures.

Notes

Note 1: AIX and the IBM 6150 RT/PC

AIX is IBM's version of UNIX. This runs mainly on IBM's RS/6000 range of machines. The 6150 RT/PC was a forerunner to these computers.

Note 2: VSAPL and APL2

VSAPL and APL2 are IBM products. I used them on a mainframe under the VM/CMS operating system. APL2 is also available for other operating systems, including other mainframe operating systems, OS/2, AIX and SunOS. A Windows version is currently in the beta testing phase.

Note 3: APL*Plus

APL*Plus is a family of APL interpreters created by STSC (originally a time-sharing company that no longer exists). The UNIX and DOS implementations are now in the hands of APL2000, while the mainframe product now belongs to Manugistics.

Note 4: OpenWindows

OpenWindows is Sun's implementation of the X Windows environment of the Open Look window manager. This was the standard Sun user environment until Solaris 2 became CDE compliant. It is still an option provided with the Solaris 2 distribution.

Note 5: UNIXes where Dyalog APL is supported

Dyalog APL is available for many UNIX variants. I have used it running under AIX (IBM), SCO Unix, Dynix (Sequent), SunOS 4 & Solaris 2 (Sun), Xenix (SCO), and I know that it is available for IRIX and HP-UX. There may be other supported operating systems that I am not aware of. From the supported window managers listed in the literature provided with the review software, I gather that at least Sun (Solaris 2), Hewlett Packard (HP-UX) and Silicon Graphics (IRIX) are supported platforms for APL/M. There may be others that I am not aware of — interested parties would have to contact Dyadic Systems.

Note 6: Window managers

Just as UNIX vendors do not integrate X Windows with the operating system, so the windowing systems leaves the window manager unbundled. This means that a user may conceivably use a different window manager than those provided by the UNIX vendor (or X Windows vendor) if he prefers it. The window manager usually controls the look and feel of the interface, e.g. the window adornments and the buttons on the title bar, the actions performed by the three mouse buttons, drag/drop and copy/cut/paste implementation, the mapping of the viewable to the logical screen, icon positioning and more. Many free window managers are available on the Internet, and for a taste of these I recommend looking at "Window Managers for X", `http://www.PLiG.org/xwinman/`.

Note 7: Shell scripts

Shell scripts are the UNIX equivalent of the batch file in DOS. However, unlike `*.bat`, the shell supports a very rich and complete programming language of its own. The shell itself is the command interpreter (command.com in DOS) and will, even on the command line, allow the competent user an amazing amount of power and flexibility. In addition, most versions of UNIX come supplied with more than one shell and more are available on the Internet, each with its own features. All UNIXes are supplied with a Bourne shell and the Korn shell is becoming very popular. Other well known shells include the C shell, bash, rsh and more. Possibly the most interesting is dtksh, the Desktop Kornshell (included with all CDE compliant systems) which provides a full GUI windowing interface.

Note 8: Symbolic links

Symbolic linking is one of the few UNIX file system capabilities that Windows NT has not managed to copy. This feature is nothing more than a pointer in the file system, but can be extremely useful. On my system, for instance, the main disk is rather full and the APL/M installation would not actually have fitted in

its logical place (in the partition mounted as `/usr` as `/usr/mdyalog`) although this was where I wanted to make it available. I did have more than enough space on another disk's partition, which is mounted as `/export/home`, so I created the directory `/export/home/mdyalog`, symbolically linked this to `/usr/mdyalog`, and just installed in this last directory. Problem solved!

Note 9: CDE

The Common Desktop Environment, or CDE, is the result of an agreement between many of the major UNIX vendors to provide a common "desktop" as a standard. Among other things this includes the use of `dtwm` as the window manager (derived from `mwm`, the motif window manager), `dtlogin` to provide a graphical login screen, `dtksh` (Desktop Kornshell) as an optional shell for GUI interfaces and a variety of desktop tools (editors, shell interface windows, email readers, etc.). All these features are (of course) changeable by the user (except that only the superuser, root, has the ability to change the login interface).

Note 10: Workstations

The term "workstation" was in use in the UNIX community to denote a powerful desktop machine running UNIX long before PCs started to be described as such. I use it strictly in this form. For me, a PC is a PC, and will forever remain a PC, even if running a UNIX or UNIX-like operating system. A workstation would typically have at least a 19 inch monitor and a decent frame buffer (advanced graphics card to the PC users), a fast processor, lots of memory, and a build quality that PC users can only dream about. Old workstations (like my SPARCstation 2) may be much slower than current PCs, but are still more than capable of doing a good day's work and will probably run problem-free for another decade at least — now who can say that about the 80286 or even '386 based machines that were current when the SPARC 2 was new?

Note 11: Component files

Most APLs provide some form of component file system. These files are the most natural and convenient way to store data for APL applications.

Note 12: Linux

This is a free UNIX-like operating system named after its creator, Linus Torvalds. Development was done across the Internet and progressed very rapidly. It is now a stable system which will run on many platforms and use the system resources efficiently. It is usually distributed with a free X Server from XFree86 and much software from the Gnu stable.

Note 13: Automounting

When sharing file resources with other machines across a network, it is often not desirable to have the other machine's disks mounted continuously. Many UNIXes now provide an automounter, which will mount resources transparently when required. Probably the most obvious (but by no means only) use for this is to mount the user's home directory (where his/her personal files are stored, including all initialisation scripts) so that the user will have the same environment on any machine on the network.

Note 14: NFS

The Network File System, NFS, originated from Sun Microsystems in the early 1980s. Essentially (and very simplistically) this is a protocol whereby two UNIX machines may share resources, especially file/disk resources, while maintaining an acceptable level of security. It has also been ported to PCs (Sun has a product called PC-NFS) to allow cross-platform resource sharing as well.

Web Site References

These may be useful for those who want more information:

APL	
APL and J Home Page:	http://www.acm.org/sigapl/
APL2000:	http://www.APL2000.com/
Dyadic Systems:	http://www.dyadic.com/
Vector:	http://www.vector.org.uk/
The Waterloo APL Archives:	ftp://watserv1.uwaterloo.ca/languages/apl/Welcome.html
Sun	
Sun Microsystems:	http://www.sun.com/
Sun User Forum:	http://www.sunuserforum.org/
SunWorld:	http://www.sun.com/sunworldonline/index.html
Linux	
Debian GNU/Linux:	http://www.debian.org/
GNU (Free Software Foundation):	http://www.gnu.ai.mit.edu/
Window Managers for X:	http://www.PLiG.org/xwinman/
XFree86(TM): Home Page:	http://www.sunsite.doc.ic.ac.uk/XFree86/
The Author	
The Author's web site:	http://www.khamsin.demon.co.uk/
The Author's email address:	Bob.Hoekstra@khamsin.demon.co.uk

THE EDUCATION VECTOR

ZARK Newletter Extracts

introduced by Jon Sandles

The ZARK APL Tutor is a computer-based tutorial for learning APL for the PC. If you bought the tutorial package you would also receive free subscription to the ZARK APL newsletter for one year. This quarterly newsletter provided additional tutorial exercises and an excellent series of APL crosswords.

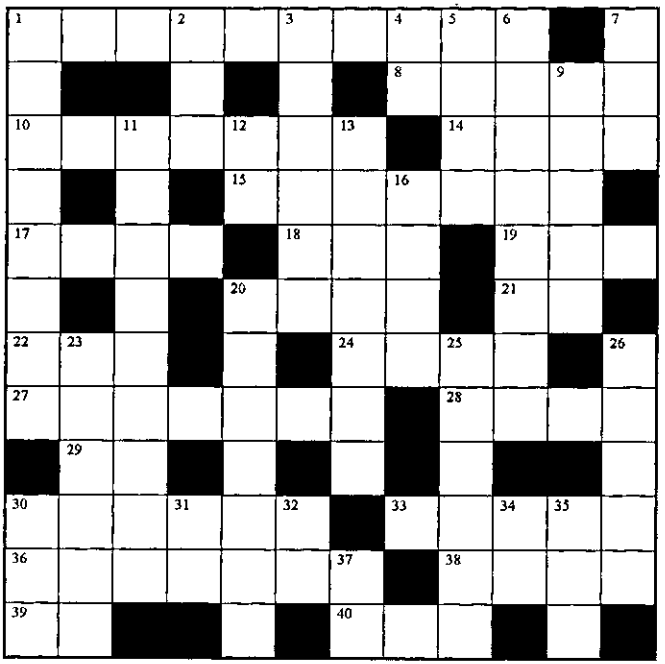
Vector has been granted permission to reproduce extracts from the ZARK APL newsletter and we have decided to run a series of re-prints in every issue. We currently have issues from 1989 to 1994 and we will reprint crosswords and exercises that are still relevant. We welcome letters with new solutions and comments to the problems presented (maybe in J?) and we will publish the solutions that ZARK published in subsequent issues.

The training package was an interactive tutorial, teaching you about all aspects of APL programming. One of the advantages of APL being interpreted is that it is effectively already an interactive environment for learning the language. ZARK took this a stage further by providing a number of tutorials explaining different parts of the language and then prompting for the trainees' attempts at solving a number of simple problems. Often when the APL approach to a problem is non-intuitive ZARK would explain the APL solution by simulating the arguments that the APL creators would have had when designing the language. This is both enlightening and amusing (and more than likely involves a large slice of artistic licence).

Although the tutor was based around the APL*PLUS II interpreter it provided a good grounding for any APL dialect. When I worked at Nestlé, the tutor was used for training both mainframe and PC programmers and seemed to be reasonably well liked (I happened to just miss out — I was the last graduate to be trained by the more traditional "sit down and read Gilman and Rose" method).

The newsletter was equally as popular, its arrival being greeted by something resembling enthusiasm (a reaction the arrival of *Vector* rarely received).

In this first reprint we present the Crossword from issue 1989:4



Across

1. The indices of the elements in the vector *R* containing integers between 1 and 9.
8. *MV* is a matrix of monthly totals. Annualize them.
10. The number of dimensions in *TABLE*.
14. Identify the elements of the matrix *N* not found in the list *L*.
15. Identify the elements of the integer matrix *ME* that match the elements of the corresponding matrix *M*, after scaling (dividing) *M* by the scalar *N* and truncating fractions.
17. The available programs.
18. It's simpler than $2 \times A$, and gives the same value.

19. Identify the elements of *D* that are smaller than the corresponding elements of *K*.
20. Return all or none of the columns of *M*, depending on whether *BR* is 1 or 0.
21. The pointless inverse of the pointless expression `-⍳0`.
22. The rankings of the elements of the vector *V*, 1 for largest, 2 for next largest, and so on.
24. `(⍷'⍷''''''BT''''',⍤⍴''''BT''''''')/'~'`
27. The end-of-year balances of an investment amount *A*, given the annual interest rates in the vector *I*.
28. The area of a circle having radius *R*.
29. `,2 1⍴2 1⍟2 1⍴'TUVW',[.21]'1234'`
30. `¯2 ¯7 ¯12 ... ¯37 ¯42.`
33. Round each number in the matrix *N* to the nearest integer.
36. Draw one card from a deck of playing cards and then put it back. Do this 100 times.
38. `,⍴(25,⍴V)⍴V`
39. `-/32 1 2 3`
40. Five fives

Down

1. Are the elements of the vector *A* in ascending order?
2. *A* is a non-negative scalar. Return 1 if *A* is positive, or an empty vector if *A* is zero.
3. Throw it all away!
4. The indices of the rows of a matrix with *M* rows.
5. The number of elements of the vector *V* not in the matrix *M*.
6. *R* is a Boolean matrix that flags the non-zero elements of *D*. Return the quotient *N*÷*D*, returning 0 whenever *D* is 0.
7. `1 1⍴3 4 ⍴'MY  32NDBOLT'`
9. Are any of the elements of the matrix *N* negative?
11. Multiply each element of *V* by 250, replicate them *N* times each and add the scalar *T* to the result.
12. A logical name for a logical array of rank 2.
13. Are there exactly *E* 1s in the logical vector *BIT*?
16. The value to be subtracted from the scalar *AMT* to leave just its fractional portion.
20. If *BV*`1←V<100`, what expression can you insert between the brackets in `V[]←0` to zero out the elements of V less than 100?
23. The indices that will re-order the vector *V*, moving the values above 10 to the front of the vector and the values below 10 to the end.
25. The cosine of 45 degrees.
26. All but the last two elements of the vector *NV*.
30. Re-order `⍳3` randomly.
31. The sign of zero.
32. `(5×2*3)++/⍳10`
34. `5×12-9*6≠6`
35. The total for each row of the matrix *M*.
37. `+/3 4*2`

J-ottings 15

by Norman Thomson

And now for something totally logical — meaningless maybe, but logicians don't give a toss about that! I am indebted to Eddie Clough for setting me off on a trail through one of the more entrancing by-roads of J.

This piece is about sorties into *sorites*, which, in case you didn't know, is a word which means a string of related premises, for example:

> *No Vector reader eats porridge.*
> *All Englishmen are Vector readers.*
> *No devolutionist ever declines porridge.*

Sorites are much associated with Lewis Carroll who published a stream of delightful nonsense in the format, the gist of which is: given the truth or falsity of a compound premise such as "p1 and p2 and p3" where p1, p2 and p3 are simple premises such as those above,

(i) what combinations of truth and falsity in the elementary propositions which underlie the premises are required to legitimise the sorites?

(ii) what further true premises can be made?

A nice feature of J is `b.` which, in spite of being called an adverb, takes as its argument an integer in the range 0 to 15 (actually _15 to 15 with modulo 16 conversion) and produces the corresponding logical binary verbs. The most familiar binary verbs are

```
and=.1 b.
or=.7 b.
imp=.13 b.   NB. implies
eq=.9 b.     NB. equals
xor=.6 b.    NB. exclusive or
```

complemented with the unary operator

```
not=.[:-.]
```

The argument of `b.` when rendered as a four-bit binary number can be read as the truth table of the corresponding verb, interpreting 0/1=false/true, and reading the truth table entries as corresponding to the proposition value pairs 00,

01, 10 and 11. (The `b.` verbs can also be used monadically, in which case the result is the same as for a left argument of zero in the binary case.)

The complete set of sixteen logical operations is given in the following table in which ' represents "not", ^ represents "and", and alternative meanings are given in some cases, for example 7 and 11.

```
 0 False              4 a'^b                    8 nor, a'^b'     12 a'
1 and, a^b       5 b                   9 eq, =          13 imp, a=>b,b'=>a'
 2 a^b',(a=>b)'   6 xor, not=           10 b'               14 a=>b', b=>a'
3 a             7 or,b'=>a,a'=>b  11 b=>a,a'=>b'  15 True
```

Observe the distinction between the verbs `False` and `True` and the values false and true. The former always return the latter, regardless of their arguments.

Symbolically the premises in the above sorites, in terms of elementary propositions are:

A: is a Vector reader
B: is an Englishman
C: favours devolution
D: eats porridge

and using this condensation, the sorites can be summarised

A imp not D
B imp A
C imp D

Write the compound proposition

```
s1=.(A imp(not D)) and (B imp A) and (C imp D)
```

Now ask question (i) above, namely — what combinations of true/false values of A, B, C and D are consistent with the truth of `s1`? At this point digress to consider a verb which generates a list of the first `i.2^y.` binary numbers:

```
   binlist=.#:@i.@(2&^)
   binlist 3
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

For presentation purposes it is more convenient to transpose this, hence the verb `bintab`:

```
   bintab=.|:@binlis      NB. binary numbers in ordered columns
   ]b4=.bintab 4          NB. list of (i.2^4) as 4-digit binary numbers
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

`b4` is not only a representation in binary form of the integers `i.16`, it is also an array whose columns in order are the truth tables as described above for the sixteen logical verbs, and whose rows correspond to four propositions, all of whose combinations of truth and falsity are represented in the array. Remembering that `b.` provides all possible logical verbs, the truth of any logical operation involving two propositions can be obtained by selecting two rows, and applying the required operation. For example the logical "and" of the first and third rows is given by

```
   (0{b4) and 2{b4
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1
```

A nice device is to define

```
   A=.0&{
   B=.1&{
   C=.2&{
   D=.3&{
```

so that the above operation is simply `A and C`, and `s1` above now becomes a directly executable verb:

```
   s1 b4
1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0
```

which can be extended to include the proposition combinations by using a hook with `,`

```
   (,s1) b4
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0
```

It is useful also to be able to select only those columns for which s1 is true, or alternatively only those for which s1 is false. Recalling that s1 is a verb, this suggests the following adverbs to qualify such premise verbs:

```
   true=.1 : '#"1~x.'          NB. adverb gives cols for which x. true
   false=.1 :'#"1~(not x.)     NB. adverb gives cols for which x. false

   s1 true b4
0  0  0  1  1
0  0  0  0  1
0  0  1  0  0
0  1  1  0  0
```

which says in substantive terms that s1 is confirmed provided Vector readers neither favour devolution nor eat porridge (cols. 4/5), porridge eaters are neither English nor read Vector (cols. 2/3), and there are no English devolutionists.

Question (ii) concerns what further information can be deduced. Confining attention to pairs of propositions, the original three premises connected AD, CD and AB respectively. In order to make statements about the other three proposition pairs, namely AC, BC and BD (glimmerings perhaps of artificial intelligence here!), first convert a string such as 'AB' into numerical indices, and then apply these to the columns of s1 true b4. Upper-case characters are selected by

```
   caps=.(65+i.26){a.
```

and the conversion and indexing processes by

```
   ones=.|:@({~caps&i.)
   (s1 true b4)ones 'AC'
0 0
0 0
0 1
1 0
1 0
```

Now determine which of the columns of the truth table for two propositions are represented in the above columns:

```
   ]bin2=.binlist 2
0 0
0 1
1 0
1 1

   bin2 e.(s1 true b4)ones 'AC'
1 1 1 0
```

Converting this binary number to a decimal integer gives the index of the logical verb which applies to A and C, which can be incorporated into a verb which locates the truth table number for a given pair of propositions:

```
   ttno=.(2&#.)@(bin2&e.)@ones
   (s1 true b4) ttno 'AC'
14
```

that is A=>C', or in words, no Vector readers are devolutionists. For the record, the other conclusions are that Englishmen don't favour devolution either, nor do they eat porridge.

It is a minor irritation that the array b4 has to be set up in advance, since as far as I am aware, there is no mechanism in J which allows the body of a verb to be analysed as a text string. One possible way around this is to duplicate the premise as a character string and count the number of capitals:

```
   pno=.+/@(caps&e.)
   pno '(A imp(not D)) and (B imp A) and (C imp D)'
4
```

Here is another sorites:

There is no sense in this article.
Whatever is not sensible is intolerable.
J users are invariably tolerable.

for which the elementary propositions can be represented:

A : uses J
B : is this article
C : is intolerable
D : is sensible

The compound premise defining the above is:

```
   s2=.(B imp(not D)) and ((not D) imp C) and (A imp (not C))
```

from which it can be concluded that:

```
   (s2 true b4)ttno 'BC'     NB. identify truth table for A/B given s2
13
```

which is perhaps better not translated!

APL2000 User Conference Nov 2-5 1997 Sheraton World Resort, Orlando, Florida

reviewed by Ray Cannon

This was the second year that the APL2000 User Conference was held in Orlando, and the first that I have attended. As a regular attendee at the annual International APL conference (which I sadly missed this year) I was very impressed by the organisation, contents and location, and would like to thank all the hard working individuals at APL2000 who made it all possible.

The conference was held over 4 days, 2 of which were given over to training sessions, and the other 2 to talks, reviews, "product announcements" and presentations.

At registration, everyone was given a folder complete with time-table, conference attendees list, notes for each session (often with complete photo-copies of the overhead projection foils) and also a very nice fleece jacket with an APL2000 emblem on it. In addition to beta copies of APL+WIN (version 3.0) and floppy disks with additional material from the sessions, we were all given a box of six "self help" audio tapes by Barry Green (who was one of the invited speakers).

The overall level was very high, and I hope to be able to attend again next year.

Training Sessions

Ray Polivka led the "Introduction to APL" course. This was a 6-hour course in two parts designed for persons with no previous APL knowledge.

Rick Butterworth led the "Introduction to Programming in APL+Win" course. This was another 6-hour course but for APL programmers "new" to Windows.

Gary Bergquist led the "Computer Based Training for APL" course. This was a 3-hour introduction to ZARK's APL Tutor.

Gary also led the "Intermediate APL Programming" course. This was another 6-hour, two part course, and was designed for APL programmers who wanted an introduction or review of some of the more advanced features of APL. (Nested arrays, Control Structures, and performance tuning.)

Eric Lescasse led the "Intermediate APL+Win for Applications" course. This was another 6-hour two part course designed to show how to bring all the elements (control, forms, functions, etc.) of a successful Windows application together.

Finally, there was an "Intermediate Programming in APL+Win" course given by some of APL2000's staff. This course demonstrated some of the Windows 95 controls included in APL+WIN.

As can be seen from the above, the width of training sessions available was very great, and the courses were given by programmers of (APL) world renown. I was however unable to make up my mind which I should attend, so played "hookey" from school, and sneaked off with my daughter to visit Disney's "The Magic Kingdom" to see Mickey Mouse on the Sunday, and then to Universal Studios for the rides, and then on to the Hard Rock Cafe on the Wednesday.

Monday 3rd November

Following an early start at 8am of Registration and Refreshments, Eric Baelen (the President of APL2000) gave his welcome address. He described APL2000 and its parent company LEX2000, and its roots in GE (General Electric). APL2000 has offices in six states, and in Nottingham UK. It employs 34 people (11 at its startup in 1995) and is a "full service APL Vendor", a company "Built on APL".

We were warned that last year several laptop computers were stolen during the conference. One attendee suggested that we could "fit your laptop computer in the safes in your room" to which another attendee replied "I've already got my shoes in my safe". I was a bit puzzled about this, until I realised that in addition to the safes being fire-proof, they were also air-tight.

Eric then went on to give a preview of APL+WIN version 3.0

APL+WIN Version 3.0

Initially Eric described the new features in release 2.0 which came out since the last user conference. These included enhancements to Trees and Lists (Drag and Drop, In place editing), RTF controls, OCX controls, and APL2 support enhancements.

Due for final release in February 1998 (along with APL DOS 6.0) we, the attendees, had already been given beta copies of version 3.0.

This release's new features include :

- TCP/iP interface (□*NI*)

- APL+Win as an Inter/Intranet server via ODBC (APL+ODBC)
- Media Player
- Numerous Other GUI Enhancements

To my mind, one of the most useful features was also one of the simplest, access to the time stamp stored within each function.

Quote of the session: "If `)OFF YES` suppresses the conformation prompt, what does `)OFF NO` do?"

Ryder Systems

After the morning coffee break, Scotty Elmslie of Anderson Consulting described the Financial Modeling Package (FMP) and "OPTIONS" built on APL that Ryder Systems use.

Ryder Systems are a large (120 locations) company whose primary business is the leasing and maintenance of vehicles with all the associated logistics. (The well known yellow "Ryder Truck Rental" part of the business was sold and now forms no part of their current business.)

FMP was developed for the mainframe back in 1978 and is a multi-dimensional database with model data and programs in "keyed" files. Scotty went on to describe briefly how the application/APL usage evolved from 3 applications and 1 APL programmer with 10 users in 1975 to many mainframe and PC applications with 6 developers and over 700 direct users (500 users of OPTIONS).

OPTIONS is a package that allows the user to price a leasing and maintenance contact right through to producing the legal documents and setting up new customers for billing. I was impressed by its clarity and ease of use. Quote of the session: "Used Cars Sales Persons using an APL system to price the product in front of the customer."

(Note. At one time OPTIONS used "Power Builder" to print the legal reports, but their users were not happy with this and now printing is done by MS Word. Moral, use the right tool for the job, not necessarily APL.)

Scotty went on to explain why APL was used and what challenges APL programmers would have to face in the future.

Breakout Sessions

The bulk of the conference was 6 sets of 3 concurrent sessions (a total of 15 different sessions, some of the more popular ones being repeated.) The sessions I was unable (unfortunately) to attend were:

- Brian Chizever (APL2000) on "Windows Design Guide";
- John Walker (APL2000) on "Frequently Asked Help Desk Questions and Answers";
- Gary Bergquist (ZARK Inc.) on "Tune-up your APL from an Actuarial Perspective";
- Jeff Pedneau (Softmed Systems) on "Code File Management";
- Rick Butterworth (Peak Software Inc.) on "Business Graphics and APL+WIN";
- James Wheeler (various) on "Object-Based Applications Architecture";
- Colyn Phillips (APL2000) on "Difference between Print and Online Documentation";
- APL2000 Staff in an Open session.

On Monday afternoon, after a well earned lunch, I attended Mark Osborne and Michael Steiner's (of APL2000) "Introduction to the Network Interface". This was an excellent introduction to MS Windows sockets and APL+WIN and APL+UNIX new ⎕*NI* system function, and left me wanting to try it out. Quotes of the session:

> "Save your workspace before you do anything"
>
> "Just like any other standard, everybody has got their own"
>
> "If you are talking to yourself, you want to use APL"

After a coffee break, I attended James Wheeler's excellent talk "Internet Client Application in APL+WIN" which came with some free software (an OCX of custom internet tools) and the quote "Kids, it's safe to try this at home". Amongst the gems of wisdom James gave out was this on ASCII vs Binary in FTP transfer: "ASCII will do you favours for which I was often NOT very grateful".

James's session was immediately followed by Michael Steiner and John Walker: "Making your APL+WIN Application into an ODBC Server". This session also came with software, a beta copy of a 32-bit DLL and MS ODBC Driver. With this software we were shown how ODBC client (non-APL) software using standard SQL could access data in proprietary (APL) systems. I considered that this session had the most far-reaching consequences of all the ones I attended. (Quote: "ODBC is a standard, which means each implementation is different.")

The final talk on Monday was given by Barry Green of the Pinnacle Group entitled "Selling APL Applications". Barry's talk was most interesting and enjoyable, as well as being the most interactive. (I am also enjoying his audio tapes.) Barry — with 25 years' experience in selling, he started out selling shoes — proved to us that we are ALL Sales Persons, whether selling a product (an application written in APL), a language (APL), or ourselves (APL programmers).

Barry's gems:

> "The most important thing about selling is the close."
>
> "20% of all prospects produce 80% of the sales, the trick is to know which to avoid."
>
> "Good sales persons control the environment."
>
> "Customers buy, Be prepared, KNOW."
>
> "Know your customer's needs, SOLVE your customer's problem."
>
> And how to close a sale? Say "We need to know right now."

Barry gave a host of interesting stats including "87% of all persons who ask for literature, expect to purchase (but not necessarily from you!)".

Tuesday 4th November

Following Monday's Banquet and Entertainment, the early start on Tuesday morning was too much for me. I missed the "Continental Breakfast" and "Opening Remarks" (I did however manage to get a full American breakfast in the hotel's restaurant) and arrived just in time for the invited speaker Rick Butterworth's (Peak Software Inc.) talk on "Data Warehousing."

Now "Data Warehousing" was a new term for me, and I am still a bit unsure of what it is offering, so I will supply a few quotes from Rick's accompanying paper/slides.

Why Build a Data Warehouse?

- Because we can
- Lets you see your Data
- Adds Value to a Product or Service
- Competitive Advantage

Data Warehouse Goals:

- Publish Quality Documented Data
- High Speed Personal Access
- Multi-Dimensional Viewpoint
- User Tools to Analyze, Organize and Present

"The first Data Warehouse goal is to publish usable data. This is more than extracting and dumping data to the Warehouse. It means the data is cleaned up, verified, and quality control tested before being released for use."

This all seemed a bit "old hat" to me and jargon for the sake of jargon, until I saw the flyer for

> "Highlander" — A Data Mining System for On-Line Analytical Processing, a product from Peak Software Inc.

and then the penny dropped.

Quotes of the session:

> "APL is the only language that lets you walk all over your data."
>
> "Data Warehousing justifies its own platform."
>
> "The real cost in cleaning up the data."

After the mid-morning break, I attended Bill Rutiser's (APL2000) session on "OCX's (and other things) Exposed".

Bill's talk cut through the jargon (OLE, OLE2, COM, DCOM, OLE again, COM+, Compound Documents, Automation, Controls-OCX, Controls-ActiveX ... "more names than concepts") and one of his slides was even entitled "Names changed to protect ..."

Bill's session was very good, but a bit like a Chinese take-away meal, very satisfying at the time, but half an hour later I felt I could consume it again.

Quote of the session (describing the behaviour of an Object Instance): "They do something when PREvoked."(invoked).

Tuesday Afternoon

Brian Chizever's "Interfacing to External Procedures" would have been better preceding Bill's earlier session rather than following it as it put OCX into context, but that did not detract from its content.

Brian provided an overview of the methods available from APL2000 of accessing the outside world. He covered □*CMD*, □*CALL*, DDE, □*NA*, □*WCALL*, Custom Messages, □*NI*, VBX, and OCX/OLE.

□*CMD* — Run a DOS command
Avoid using it as it:

- flashes a DOS window on the screen;
- bypasses security limiting access to DOS subsystem;
- APL may not wait for completion;
- and there may well be an API call available.

□*CALL* — Call a machine language routine
Very good if you have the machine code but suffers from several problems:

- difficult (for APL'ers) to write;
- machine code differs between APL+ products;
- slow to run the first time it is called, but much faster on subsequent calls.

DDE — Dynamic Data Exchange
Limited to Windows 3.1 where it is the standard method of communication between applications. It can be quite difficult to use.

□*NA* — Name Association
Executes a command in a DLL (Dynamic Load Library). Works on all Windows platforms, but is hindered by the differences in 16- and 32-bit DLLs used by Windows 3.1 and NT. Also it does NOT support filters and callbacks.

□*WCALL* — Windows call
Similar to □*NA* in that it executes a command in a DLL, but it *does* support filters and callbacks.

Custom Messages
This method uses □*WCALL* to pass information between two of *your* applications running on the same machine.

⎕NI — Network Interface
This is a high-level Winsock interface (the standard Windows TCP/IP interface driver), and allows communication between machines and distributed processing (client/server).

VBX — Visual Basic Extension

OCX — OLE Control Extension (Active X)
These two methods are similar. Both work on Windows 95, but NT will not run VBX and Windows 3.x does not run OCX (unless it is also running Win32s). Add new classes to ⎕WI, each with properties, events and methods. (See Formula One below.)

I found this session very useful,as it helped put all these terms into context.

Final Breakout Session

Following the mid-afternoon break was the final breakout session, Eric Lescasse's second run of his well attended session on "Formula One OCX and APL+Win".

Formula One is an Excel-like spreadsheet control. Note that unlike Dyalog APL/W, APL+Win does not have its own inbuilt grid object. Eric demonstrated that the addition of Formula One more than made up for the lack of an inbuilt grid control, giving in addition to a grid object, the ability to load and save data as Excel spreadsheets or HTML documents, and a WYSIWYG print preview. I was very impressed by Formula One, and thought it would make a very useful addition to an APL+Win programmer's tool box.

PackageWorks
The final invited speaker, Richard Krafchin, gave a demonstration of his company's fully integrated system. This system (entirely written in APL) encompassed all the work done by computer in his company from messages to order processing to payroll to diary to invoice production.

This was followed by Eric Baelen's closing remarks.

My thanks once again to Eric and his team (with a special thank you to Sonia Beekman, the Conference Coordinator). I found the 1997 APL2000 User Conference in Orlando a most useful and enjoyable experience. Thanks.

APL as a Tool of Thought X
Hoboken, NJ, January 31, 1998

reported by Ed Shaw

It was a brisk and beautiful day for the Tenth *APL as a Tool of Thought*. Held at Stevens Institute of Technology in Hoboken, New Jersey, at the top of a bluff overlooking the skyscrapers of midtown Manhattan, NY/SIGAPL could not have selected a better venue for this latest in a series begun in 1983. As has been the tradition, nearly every speaker submitted a paper to be published in the proceedings.

The day began with a leisurely cup of coffee and a Danish pastry or NYC bagel over which attendees could greet old friends, make new acquaintances, and peruse the proceedings. The opening plenary session contained two surprises: **Adin Falkoff** was invited from somewhere in the depths of retirement to present the Iverson Award to **John McPherson**, APL's elder statesman. McPherson is a handsome eighty-eight year old gentleman with a ruddy face and twinkle in his eye who was IBM's first technical vice president. He brought Ken Iverson (who was also present) to IBM's System Research Institute, where Ken taught what was then called the Iverson Notation. John has been a strong supporter of APL, particularly within IBM, ever since it came into being. He has also provided consistent encouragement for the Tool of Thought seminars and has attended nearly all of them.

Eric Baelen gave an enthusiastic plenary talk encouraging all of us to keep the faith. He presented an impressive list of companies in a variety of businesses that have used APL as the basis for their success.

From all reports, the quality of the presentations, of which there were fourteen or so, in three parallel sessions, was generally excellent.

Of those I attended, I was particularly impressed by what **Richard Krafchin** has built for the five person promotional advertising company that he and his wife run. The functionality is powerful — everything they need to administer their operation from recording phone messages, planning projects, accessing historical costs, and maintaining the current accounts. On top of that Rich has experimented with the user interface to make it as simple, easy to use, and attractive as someone in advertising might expect. Who needs MS Office, Lotus Smart Suite, or an Oracle database? Not he. He has built it all with APL.

Eric **Iverson** gave a clear and straight-forward presentation of object oriented programming (Classes are collections of data and methods. Objects are an instance of a class. Methods are functions.) and demonstrated how it will be implemented in J's next release in March. Will others follow suit?

Linda Alvord combined her artistic, mathematical, and programming talents in J to do what she calls *Derivatives for Dancing* and what might also be called Computational Origami. Writing in J, she created figures such as a boat or a fish and moved them along a curve so that the figure used the derivative of the function/tangent to the curve as its base, causing the figure to appear to move in a natural fashion along the path of the function.

Now that other interpreters such as Java have become accepted as legitimate programming tools, **Fred Waid** feels that SGML and its offspring: HTML, XML, SMIL, and what have you on the Internet, offer a unique opportunity for APL to shine. With an interactive multimedia web broadcast as an example, Fred suggested that the rapid development time, computational power, and speed of APL can give APL a real competitive advantage to those willing to use it in this seemingly alien environment.

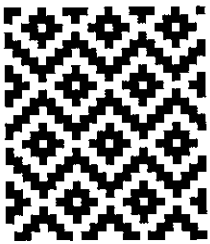
Do you think the Y2K problem is a lot of hype? There certainly is hype, but it seems to be warranted. **Clement Kent** suggests that it is even worse than people think. In an engaging presentation, he demonstrated that not only is the year a problem, but the leap day, and other days are as well. It's not clear how many programmers know the rules of leap years, particularly the third rule: century divisible by 400. Horrible events like the two aluminium smelters that crashed and burned on February 29, 1988 may occur on a broader scale. Obsolete, unsupported air traffic control computers, mainframe computers that use as a reference clocks in Geodesic satellites that will reset to zero (because of a ten-bit data store) on August 22, 1999, and Internet routers sold as recently as July, 1997 that will crash on January 1, 2000 make one contemplate going back to nature for a while.

The day ended with an entertaining talk by **Gary Bergquist**, whose programming skills are well known, but whose talent as a standup comedian is not. He had us all laughing with him as he dryly and strongly defended APL as A Perfect Language!

NB. The Proceedings which include other papers such as Armstrong and Objects, Bernecky and Compilation, Bhizever and GUI Design, Graham and APL Zero, Karman and Pension Projections, McCormick and Bayesian Modelling, and Mansour and Non-looping Mortgages, are available from NY/SIGAPL for $15.

The Computer Construction of Weaving Designs

by Keith Smillie (smillie@cs.ualberta.ca)



J algorithms are developed for deriving the weave of a piece of cloth from the instructions for setting up a loom, for the converse operation of deriving the setup instructions from the weave, and for introducing colour into the weave.

Introduction

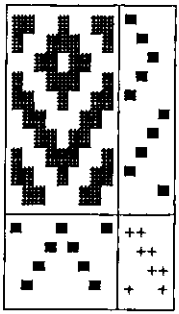
Weaving is the art of forming a fabric by interlacing threads at right angles and is performed on a frame called a loom. The threads at right angles to the weaver are the warp threads, and the threads parallel to the weaver and which are inserted between the warp threads are the weft threads. The warp threads are alternately raised and lowered to create a shed through which the weft thread is inserted. They are fastened to a warp roller at the back of the loom, pass over a back beam and through the eyes of vertical wires called heddles, and then through reeds which help keep them parallel and in the proper sequence. The woven cloth passes over a front beam and is then wound onto a cloth roller. Each row of heddles is attached to a harness, and there may be two, four, eight or more harnesses. Each warp thread passes through a heddle on any one of the harnesses. The harnesses are raised and lowered by treadles which are worked by the fingers on a table loom and by the feet on a foot loom. Harnesses may be tied together so that more than one may be operated by the same treadle. As a group of warp threads is raised or lowered by the treadles, the shuttle with the weft thread is inserted into the shed between the two groups of threads.

Drafts

The instructions for setting up the loom with the warp threads are known as drafts. The threading draft determines the order in which the warp threads are drawn through the heddles, the tie-up draft gives the connection of the harnesses, and the treadling draft gives the order in which the treadles are used. The resulting design is known as the weave draft. A cloth diagram is a rectangular

display of these four drafts with the weave draft in the upper left, the threading draft in the lower left, the treadling draft in the upper right, and the tie-up draft in the lower right.

The cloth diagram for an upper left corner of the design given at the beginning of this paper is the following:



The threading draft shows that the second and eighth warp threads pass through heddles connected to the first harness, the third and seventh threads pass through heddles connected to the second harness, etc. The tie-up shows that the first treadle is connected to the first and fourth harnesses, etc. The treadling draft shows that the treadles are used in the order first, second, third, etc. Dark squares in the weave draft indicate the visibility of the corresponding warp threads.

Constructing the Weave

For computational purposes the various drafts may be conveniently represented by boolean tables. Thus the cloth diagram of the previous section may be represented as

```
+-----------------------+
|1 1 0 0 1 0 0 1 1|1 0 0 0|
|1 0 0 1 1 1 0 0 1|0 1 0 0|
|0 0 1 1 0 1 1 0 0|0 0 1 0|
|1 0 0 1 1 1 0 0 1|0 1 0 0|
|1 1 0 0 1 0 0 1 1|1 0 0 0|
|0 1 1 0 0 0 1 1 0|0 0 0 1|
|0 0 1 1 0 1 1 0 0|0 0 1 0|
|1 0 0 1 1 1 0 0 1|0 1 0 0|
|1 1 0 0 1 0 0 1 1|1 0 0 0|
|0 1 1 0 0 0 1 1 0|0 0 0 1|
+-----------------+-------|
|1 0 0 0 1 0 0 0 1|1 1 0 0|
|0 0 0 1 0 1 0 0 0|0 1 1 0|
|0 0 1 0 0 0 1 0 0|0 0 1 1|
|0 1 0 0 0 0 0 1 0|1 0 0 1|
+-----------------------+
```

where the interpretation of the 0s and 1s is apparent. In this section we shall show how the weave draft may be very simply calculated from the threading, tie-up and treadling drafts. For convenience, we shall let these four drafts be represented by W, H, I and R, representing *w*eave, *th*reading, *ti*e-up and treadling, respectively.

First let us define the logical dot product

```
both=. +./ . *.
```

which gives a value of 1 if at least one pair of corresponding items in its list arguments is 1, and 0 otherwise. Now the expression

```
R; (|:I); T=. R both |:I
```

has the value

```
+-----------------------+
|1 0 0 0|1 0 0 1|1 0 0 1|
|0 1 0 0|1 1 0 0|1 1 0 0|
|0 0 1 0|0 1 1 0|0 1 1 0|
|0 1 0 0|0 0 1 1|1 1 0 0|
|1 0 0 0|       |1 0 0 1|
|0 0 0 1|       |0 0 1 1|
|0 0 1 0|       |0 1 1 0|
|0 1 0 0|       |1 1 0 0|
|1 0 0 0|       |1 0 0 1|
|0 0 0 1|       |0 0 1 1|
+-----------------------+
```

The rows of the third table T indicate those harnesses activated by the treadles in the corresponding weft rows of the weave, and, for example, the first row indicates that the first and fourth harnesses are affected. Consequentially, the expression `T both H`, which is equivalent to `(R both |:I) both H` will give the required boolean representation of the weave draft. This may be seen from the expression `T; H; T both H` which has the value

```
+-----------------------------------------+
|1 0 0 1|1 0 0 0 1 0 0 0 1|1 1 0 0 1 0 0 1 1|
|1 1 0 0|0 0 0 1 0 1 0 0 0|1 0 0 1 1 1 0 0 1|
|0 1 1 0|0 0 1 0 0 0 1 0 0|0 0 1 1 0 1 1 0 0|
|1 1 0 0|0 1 0 0 0 0 0 1 0|1 0 0 1 1 1 0 0 1|
|1 0 0 1|                 |1 1 0 0 1 0 0 1 1|
|0 0 1 1|                 |0 1 1 0 0 0 1 1 0|
|0 1 1 0|                 |0 0 1 1 0 1 1 0 0|
|1 1 0 0|                 |1 0 0 1 1 1 0 0 1|
|1 0 0 1|                 |1 1 0 0 1 0 0 1 1|
|0 0 1 1|                 |0 1 1 0 0 0 1 1 0|
+-----------------------------------------+
```

where, for example, the first row of third table, which gives the weave draft W, shows that the first, second, fifth, eighth and ninth warp threads are visible in the first weft row.

We may define the verb

```
weave=. (2&get both |: @ (1&get)) both 0&get
```

where

```
get=. >@{
```

whose argument is the three-item list H;I;R of threading, tie-up and treadling drafts and whose result is the required weave draft. Finally a cloth diagram is given by

```
diagram=. ((weave);2&get),: (0&get);1&get
```

whose argument is the same as that for **weave**. The cloth diagram in the previous section is given by the verb **Cdiagram**, with the same syntax as **diagram**, and is given in the script file in the Appendix.

Analyzing the Weave

In this section we shall consider the construction of the threading, tie-up and treadling drafts from the weave draft. First of all, we note that if a is an arbitrary table, say,

```
1 4 1 7 1
2 5 2 8 2
3 6 3 9 3
```

then the expression <"1 |: a gives the list

```
+-----------------------------+
|1 2 3|4 5 6|1 2 3|7 8 9|1 2 3|
+-----------------------------+
```

of the columns of a, and = <"1 |: a gives the table

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

of their distribution.

For a given weave `W` the threading draft is simply its column distribution so that

```
H=. coldis W
```

where

```
coldis=. = @ (<"1 @ |:)
```

and the treadling draft is

```
R=. |: coldis |: W
```

The tie-up draft is given by

```
I=. (-.|: R) either W either -.|: H
```

where

```
either=. *./ . +.
```

is the logical dot product which gives a value of `1` if at least one item in each pair of corresponding items in its list arguments is `1`, and `0` otherwise.

We can use the expressions in the last paragraph to define the verb

```
drafts=. 3 : 0
W=.  y.
H=. coldis W
R=. |: coldis |: W
I=. (-.|: R) either W either -.|: H
H;I;R
)
```

which gives a three-item list of threading, tie-up and treadling drafts corresponding to the weave draft given as the argument. For example, if `W` is the weave draft of the example in the previous section, then the expression `drafts W` has the value:

```
+-----------------------------------+
|1 0 0 0 1 0 0 0 1|1 1 0 0|1 0 0 0|
|0 1 0 0 0 0 0 1 0|1 0 0 1|0 1 0 0|
|0 0 1 0 0 0 1 0 0|0 0 1 1|0 0 1 0|
|0 0 0 1 0 1 0 0 0|0 1 1 0|0 1 0 0|
|                 |       |1 0 0 0|
|                 |       |0 0 0 1|
|                 |       |0 0 1 0|
|                 |       |0 1 0 0|
|                 |       |1 0 0 0|
|                 |       |0 0 0 1|
+-----------------------------------+
```

We note that even though some of the rows of the threading and tie-up drafts have been permuted, they may be used to construct the original weave draft from which they were derived. Indeed, for arbitrary drafts `H`, `I` and `R`, the expression

```
W -: weave drafts W=. weave H;I;R
```

should have the value `1`.

Colouring the Weave

The introduction of colour into the weave is a very simple process if we make use of just a few of the techniques discussed by Clifford Reiter in his book *Fractals Visualization and J*. In particular, we shall use his development of the RGB colour model and the verbs for generating and viewing raster graphics.

In the RGB colour model each colour is considered to be composed of certain fractions of the three basic colours red, green and blue. For example, if all three colours are absent, the resulting colour is black; if they are all present with a fraction of 1, the resulting colour is white; and if red and green are fully present and blue is absent, the resulting colou-r is yellow.

It is convenient to represent the RGB model as a unit cube with the vertices representing the following colours: (0,0,0) black, (0,0,1) blue, (0,1,0) green, (0,1,1) cyan, (1,0,0) red, (1,0,1) magenta, (1,1,0) yellow, (1,1,1) white. There is a one-one correspondence then between any colour combination and points on or within the unit cube. Since in a raster image each basic colour is represented by one byte, each colour coordinate may be represented also by a triple of integers ranging from 0 to 255 so that, for example, the combination (255,255,0) represents yellow. Finally to generate and view the raster images we shall use the two verbs `writebmp8` and `viewbmp`.

When specifying colours for weaving designs we shall represent the eight colours corresponding to the vertices of the colour cube in the order given above by `b`, `l`, `g`, `c`, `r`, `m`, `y` and `w`, respectively. Furthermore, we shall introduce eight intermediate combinations of colours represented by `B`, `L`, `G`, `C`, `R`, `M`, `Y` and `W`, where `Y` represents (128, 128, 0). This representation is given by the list

```
Colours=. 'blgcrmywBLGCRMYW' .
```

The corresponding coordinates are given by the table `Palette` with rows `0 0 0`, `0 0 255`, ... `128 128 128`, `0 0 128`, ... The colouring of a design according to these parameters is handled by the verb `BMP` given in the Appendix. The details of this verb need not concern us, and we shall note only that its right argument is a weave draft produced by the verb `weave` and the left argument specifies the warp and weft colours as discussed in the next paragraph.

A coloured weave corresponding to specified threading, tie-up and treadling drafts is given by the ambivalent verb `front` defined as

```
front=. 3 : 0
('b';'w') BMPview weave y.
:
x. BMPview weave y.
)
```

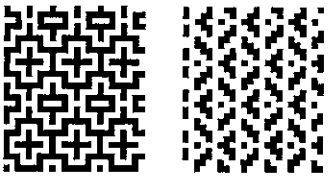
where the right argument is the three-item list `H;I;R` of the drafts. The optional left argument is a two-item list giving the warp and weft colours, and, for example, the list `'rg';'y'` specifies a warp with alternate red and green threads and a yellow weft, and the list `'b';'w'` gives the default black warp and white weft. A similar verb `back` will specify the reverse side of a coloured weave. The verb `BMPview` defined as

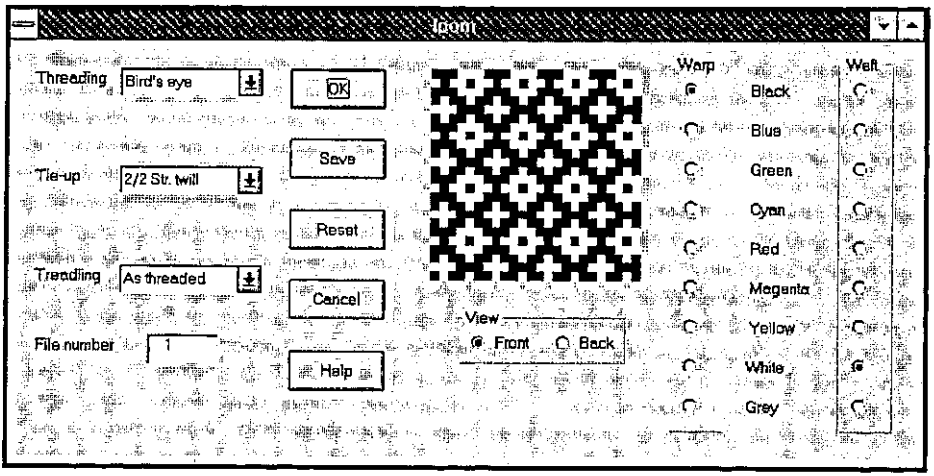
```
    BMPview=. 3 : 0
:
x. BMP y.
IMAGEsize viewbmp IMAGEfile
)
```

generates and then displays the file containing the raster image of the weave. The window size and the file name are given by the global variables `IMAGEsize` and `IMAGEname`, respectively, whose default values are given in the script file.

An indication of the influence of the sequence of colours for the warp and weft threads may be seen from the two designs given below which have the same

drafts as the one shown at the beginning of this paper which has an all black warp and an all white weft. The left weave has a warp and weft given, as a left argument to the verb `front`, by `('bw';'wb')` while the right weave has the warp and weft given by `('bwbb';'w')`.





A Windows Loom

Some of the verbs developed in the previous sections have been used in the construction of a Windows form that shows the weave resulting from any one of a number of combinations of threading, tie-up and treadling drafts and warp and weft colours. The figure shown on this page gives the appearance of the form for one of these combinations. The script file is given by anonymous ftp at `ftp.cs.ualberta.ca` in the file `pub/smillie/loom.js`. Documentation is given in the Help menu which is as follows:

The "Windows loom" permits the generation of the designs resulting from a number of combinations of threading, tie-up and treadling drafts and warp and weft colours. Selected designs may be stored as graphics files for later use. The

selection of drafts has been taken from "Weaving. A Handbook for Fiber Craftsmen" by Shirley E. Held (Holt, Reinhart and Winston, Inc., New York, 1973).

The following drafts are available and are selected from the appropriate menu:

Threading: Twill, Goose eye, Rosepath III, Cord, velveret, Broken twill, Bird's eye, Wheat

Tie-up: Plain, 2/2 Basket, 2/2 Straight twill, 1/3 Straight twill, 3/1 Straight twill, Herringbone twill

Treadling: Straight, 2 Straight, Reverse, As Threaded

Either the front or the back of the design may be viewed.

The colour of each of the warp and weft threads may selected from the following: black, blue, green, cyan, red, magenta, yellow, white and grey with the default colours being black for warp and white for weft.

The following controls are available:

OK: Generates the design for a given combination of drafts, etc.

Save: Saves the displayed design as a bit-mapped file numbered 1, 2, ..., 25 and increments the file number by 1. The file number is attached to the base file name with a default value
"c:\j303a\temp\tempxx.bmp"
where "xx" is the file number.

Reset: Resets the drafts, view option, warp and weft colours, and file number, and the design to the J logo.

Cancel: Exit program

Help: View this text.

Keith Smillie
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1
smillie@cs.ualberta.ca

References

Frey, Berta, 1975. *Designing and Drafting for Handweavers*. Collier Books, New York.

Held, Shirley E., 1973. *Weaving. A Handbook for Fiber Craftsmen*. Holt, Reinhart and Winston, Inc., New York.

Hoskins, Janet A. and W. D. Hoskins, 1981. "The solution of certain matrix equations arising from the structural analysis of woven fabrics." *Ars Combinatoria*, vol. 11, (June), pp. 51 - 59.

Reiter, Clifford A., 1995. *Fractals Visualization and J*. Iverson Software Inc., Toronto.

Appendix. Script file

```
IMAGEsize=: 256 256
IMAGEfile=: 'c:\j303a\temp\temp.bmp'

both=. +./ . *.
either=. *./ . +.
get=. >@{
coldis=. = @ (<"1 @ |:)
replace=. ] { ({&a.)@[

weave=. (2&get both |: @ (1&get)) both 0&get
diagram=. ((weave);2&get),: (0&get);1&get

Cdiagram=. 3 : 0
D=. 32 178&replace weave y.
H=. 32 254&replace 0&get y.
I=. 32 43&replace 1&get y.
R=. 32 254&replace 2&get y.
(D;R),:H;I
)

drafts=. 3 : 0
W=.  y.
H=. coldis W
R=. |: coldis |: W
I=. (-. |: R) either W either -. |: H
H;I;R
)

wdsize=. 3 : 0
empty IMAGEsize=: 2&$ y.
)

filename=. 3 : 0
empty IMAGEfile=: y.
)

front=. 3 : 0
('b';'w') BMPview weave y.
```

```
:
x. BMPview weave y.
)

back=. 3 : 0
('b';'w') BMPview -. weave y.
:
x. BMPview -. weave y.
)

BMPview=. 3 : 0
:
x. BMP y.
IMAGEsize viewbmp IMAGEfile
)

BMP=. 3 : 0
('b';'w') BMP y.
:
P=. (<"0 y.) {& ({|.(|.$y.)($&.>) x.)
Palette=: (255*#:i.8),128 128 128, 128*}.#:i.8
Colours=: 'blgcrmywBLGCRMYW'
(Palette;Colours i. P) writebmp8 IMAGEfile
)

NB. Drafts for "upper left corner"
H0=. 4 3 2 1 =/ 4 1 2 3 4 3 2 1 4
I0=. |:> 1 0 0 1;1 1 0 0;0 1 1 0;0 0 1 1
R0=. |: 1 2 3 4 =/ 1 2 3 2 1 4 3 2 1 4

NB. Drafts for main figure
H1=. 4 3 2 1 =/ 24 $ 4 1 2 3 4 3 2 1
I1=. |:> 1 0 0 1;1 1 0 0;0 1 1 0;0 0 1 1
R1=. |: 1 2 3 4 =/ 28 $ 1 2 3 2 1 4 3 2 1 4 1 2 3 4

writebmp8=: 3 : 0
:
('spal';'sbmp')=.$@>"0 x.
xsbmp=.sbmp+(i.2)*4|-sbmp
h=.524289 0,(*/xsbmp),0 0,2#spal=.0{spal
h=.(54+(4*spal)+*/xsbmp),0,(54+4*spal),40,(|.sbmp),h
head=. 'BM',,a.{~,|."1 (4#256)#:h
pal=. ,(0,"1~|."1 >{.x.){a.
bmp=. ,|.(xsbmp{.>{:x.){a.
(head,pal,bmp) 1!:2 <y.
)

viewbmp=: 3 : 0
256 256 viewbmp y.
:
wd 'pc bmpviewer closeok;pn ',y.,';'
wd 'xywh 0 0 ',(":<.x.%2.5),';cc g isipicture;set g ',y.,';'
wd 'pas 0 0;pshow;'
)
```

HTML Basics for APLers – Lists

by Adrian Smith (causeway@compuserve.com)

Introduction

This is the third article in a series, which is rapidly becoming open-ended as the HTML 'standard' races ahead. Having covered tables in Vector 14.1, I would now like to back off to something rather simpler and walk through the various kinds of indented and bulleted list. You can check back to the basics in Vector 13.4 [3] and I should repeat that all the sample code is available on the Causeway web site (www.causeway.co.uk/html.zip) in both +Win and Dyalog formats. This code may be freely downloaded and used with no restrictions.

I have continued to work with the NewLeaf 'object model', although this allows a good deal more fine control over detailed appearance than can sensibly be emulated in standard HTML. Partly this is simply for my own benefit – it means I can use the same application code to target either paper or the Internet. In +Win, it is a matter of swapping in a different function file, in Dyalog I simply switch namespace. Mostly however, it is to give the design ideas a potential lifespan of more than the next few months. HTML is evolving itself rapidly out of existence, and looks likely to be replaced (in all but name) by the Microsoft Word document format, complete with style tables and all manner of fancy formatting. By using a conceptual model built around page, paragraph and text properties I stand a chance of building something that may still be useful 5 years ahead.

Content vs Layout

As with all HTML design, you must start from the proposition that you are describing content; the browser is responsible for the layout. This is absolutely contrary to the behaviour of packages like Winword, where you control:

- the exact indent point, which is where the text will wrap to if you go on a little too long and your bullet point spills over more than one line in the way this one is doing.
- the negative offset of the bullet
- ☞ the character used for the bullet symbol

... in short you can make a complete mess just by pulling a few tab stops around on the ruler. In HTML you simply say that you want to begin an 'unordered list',

an 'ordered list', and then you provide a sequence of 'list items' which the browser will arrange appropriately. You have almost no control (yet) over the indenting and paragraph spacing which will be used, nor do you get much control over the bullet character.

This is sometimes a little frustrating, and can be got around by extreme measures such as those used on the APL98 web site (choose 'View, Source' and gasp at the huge strings of non-breaking spaces -   - which have been inserted to enforce the layout); in general it is almost always better to work *with* HTML than to fight it in this way! Your pages will be much smaller, will download proportionately faster, and will work reliably on all known browsers and platforms. If you use the 'Save as HTML' option from a standard word-processor, do have a hard look at what comes out of it before subjecting the world's modems to several kilobytes of unnecessary junk.

Simple Indents (Blockquote tag)

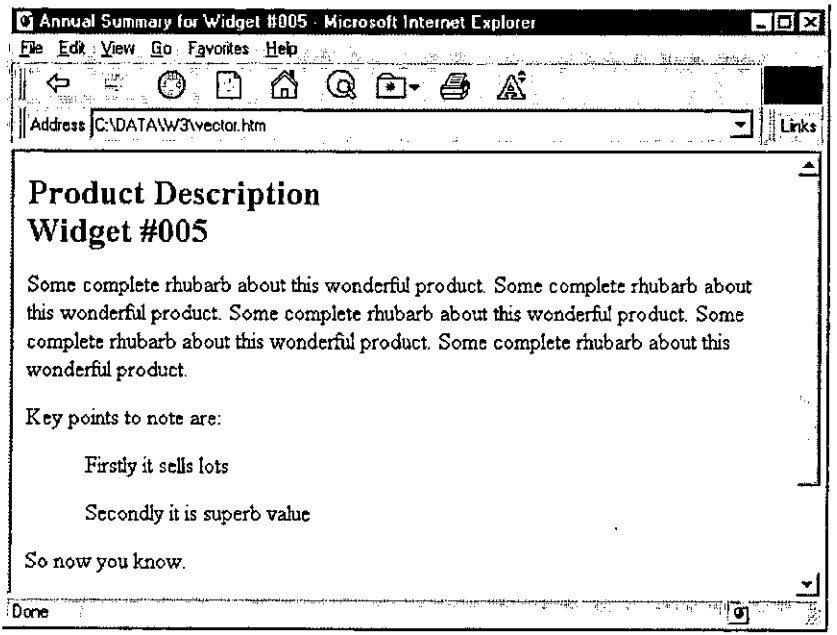
If you simply want to indent a section of text:

> This is probably a quote from somewhere else, or you might have other reasons to inset a section in this fashion.

... then the accepted approach is to surround the indented section with a pair of <blockquote> ... </blockquote> tags. You can also cheat and use the 'unordered list' tags <ul> ... but the effect is not quite the same (for example IE3 adds extra space before the list and reduces the inter-paragraph space within it) and in future you have no guarantee that all browsers will continue to show your page as you expect. Continuing the example from Vector 14.2, we could build ...

```
<HTML><HEAD>
<Title>Annual Summary for Widget #005</title></HEAD>
<BODY bgcolor="#FFFFFF">

<h2>Product Description<br>Widget #005</h2>
<p>Some comple ... oduct. </p>
<p>Key points to note are:</p>
<blockquote>
  <p>Firstly it sells lots</p>
  <p>Secondly it is superb value</p>
</blockquote>
<p>So now you know.</p>
<HR noshade size=2>
<p>&#169; Widgets International Inc<br>April 1996</p>

</body></HTML>
```

Which turns out quite nicely as:



To continue with my strategy of mirroring the NewLeaf [2] functions ...

```
    ∇ r←Indents;mat;sink
[1]   ⍝ HTML example for Vector 14.3
[2]   ⍝ Page title and product info ...
[3]    'Annual Summary for Widget #005'htmUse''
[4]    'Subhead'htmPlace 'Product Description' 'Widget #005'
[5]    htmFlow ∈5p⊂'Some complete rhubarb ... product. '
[6]
[7]   ⍝ Now some more detailed points ...
[8]    htmPlace 'Key points to note are:'
[9]    htmIndent 36
[10]   htmFlow 'Firstly it sells lots' 'Secondly it is superb value'
[11]   htmIndent 0
[12]   htmFlow 'So now you know.'
[13]   htmRule 2
[14]   htmPlace'⍝ Widgets International Inc' 'April 1996'
[15]
[16]   PG←htmClose
[17]   r←'''vector.htm'' htmPut PG  ⍝ to see it'
[18]
    ∇
```

The actual indent values are immaterial - all that we can do is note if an indent has increased or decreased, and add or remove a level of 'blockquoting' as required. We can also check the bullet character, and begin to build an 'unordered' or 'ordered' list if bullets have been specified. Most browsers take note of a starting point for the numbers, so the syntax might as well include this possibility, and we can choose from a few basic types of numbering system. What we cannot do is decide whether we want *ii* or *iii*. or even *(iii)* for our Roman sequence - that is entirely up to the browser at the moment.

Clearly, the first function in need of attention is `htmFlow`, which must check the current indent level and patch in the necessary tags:

```
    ∇ r←sty htmFlow txt
[1]   ⍝ Simple text flow, taking account of style
[2]   ⍝ Nothing useful to return here
[3]    htmUseDflt
[4]    :if 2=⎕NC'sty' ⋄ htmStyle sty ⋄ :end
[5]   ⍝ Ensure correct enclosure
[6]    :if 2>|≡txt ⋄ txt←⊂,txt ⋄ :end
[7]    htm_makelist
[8]    txt←(,/'<',¨htmΔtag,¨'>'),¨txt,¨,/(⊂'</'),¨(⌽htmΔtag),¨'>'
[9]    htm_cat txt
[10]   :if 2=⎕NC'sty' ⋄ htmStyle 'Body' ⋄ :end
[11]   r←0 0⍴0
    ∇
```

This is almost unchanged from the code shown in Vector 13.4, but note that we now call `htm_makelist` before building the paragraph. This switches the style to 'Indent' for anything other than a plain list, which has the effect that the `htmΔtag` variable is set to <li> to get correct formatting of subsequent list items. It also adds appropriate tags for the other kinds of list, so before looking at the code in detail ...

Bulleted and Numbered Lists (ul and ol tags)

I think it is safe to assume that most browsers handle bulleted lists in the same way, and that numbered lists can be indexed 1,2,3 ... or a,b,c ... or i,ii,iii and so on. A few experiments suggest that you can reliably set the starting point of the sequence, but probably you should check this conjecture before you use it.

To turn our simple indent into a bulleted list:

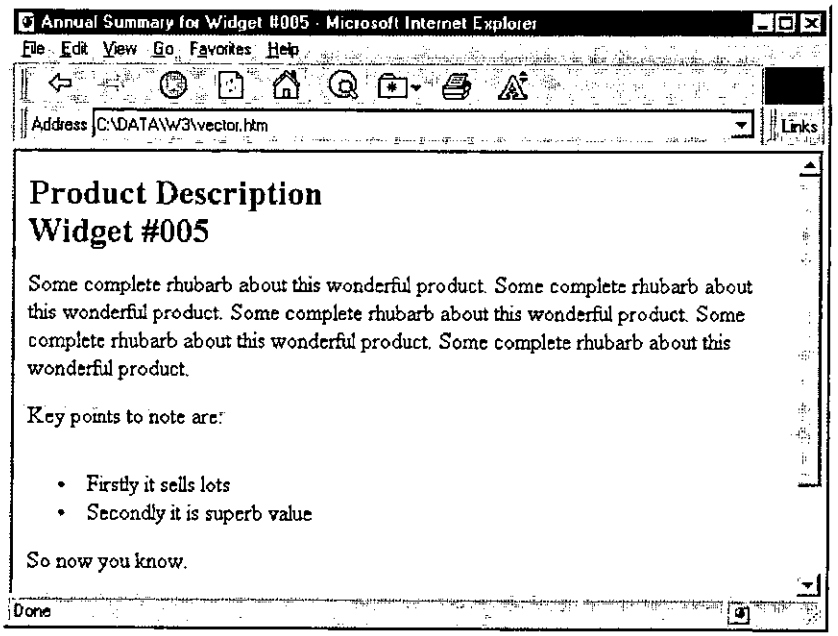
```
htmBullet '▼' ⍝ Simple bullet character (ASCII 0149)
htmFlow 'Firstly it sells lots' 'Secondly it is superb value'
```

... which should generate the HTML source ...

```
<p>Key points to note are:</p>
<ul type=disc>
 <li>Firstly it sells lots</li>
 <li>Secondly it is superb value</li>
</ul>
<p>So now you know.</p>
```

(Note that I have hardwired the bullet to the 'disc' type for consistency. Also the spaces before the <li> tags are for human readability only, they are ignored by the browser as is all redundant white space.)

This is the point where you feel like picking up a can of spray paint and daubing "S***f Bill Gates" all over the nearest plain concrete surface. Not only do Microsoft extend HTML in quite arbitrary ways, but they also fail to interpret the standard correctly! If you view this output in Netscape it looks fine; in IE3 it looks like:



Notice the extra white space above the first bullet! Pah!

The only way to prevent this is to strip the closing </p> from the end of the preceding paragraph, which adds a nasty inconsistency to the pattern of paired

tags we have been using so far. On the whole I prefer to kludge this rather than to give up the overall symmetry of the *<tag>section</tag>* pattern, so here is the code which is required:

```
    ∇ htm_strip tag
[1]   ⍝ Kludge - remove this tag if it is at the end
[2]    :if ((-1+⍴tag)↑htm_PG)≡tag,⎕TCNL
[3]       htm_PG←((-1+⍴tag)↓htm_PG),⎕TCNL
[4]    :end
    ∇


    ∇ htm_makelist;style;tag
[1]   ⍝ Process list tags, comparing with previous indent
[2]   ⍝ Effect is to step in and out of <ul> etc
[3]    :if htm_indents[1]>htm_last
[4]       :select style←↑1↓(htm_bullet∈' #@^▼')/htm_bullet
[5]        :case ' '
[6]          tag←'blockquote'
[7]        :case '▼'
[8]          htm_strip '</p>' ⋄ tag←'ul' ⋄ htmStyle 'Indent'
[9]        :caselist '#@^'
[10]         htm_strip '</p>' ⋄ htmStyle 'Indent'
[11]         tag←'ol'[⍸'#@^'⍳style]
[12]         tag←'ol type=',tag,' start=',▼htm_bulletct
[13]      :end
[14]      htm_level←htm_level,⊂(¯1+tag⍳' ')↑tag
[15]      htm_cat '<',tag,'>'
[16]      htm_last←htm_indents[1] ⋄ :return
[17]   :end
[18]
[19]   :if (htm_last>0)^htm_indents[1]=0  ⍝ Reset to normal text
[20]      htm_last←0 ⋄ htmStyle 'Body'
[21]      htm_cat (⊂'</'),¨(⌽htm_level),¨'>'
[22]      htm_level←0⍴htm_level
[23]      :return
[24]   :end
[25]
[26]   :if htm_indents[1]<htm_last
[27]      :if 0<⍴htm_level
[28]         htm_cat '</',(↑⌽htm_level),'>'
[29]      :end
[30]      htm_last←htm_indents[1] ⋄ htm_level←¯1↓htm_level
[31]      :if 0=⍴htm_level ⋄ htmStyle 'Body' ⋄ :end
[32]      :return
[33]   :end
    ∇
```

The bullets having been set/cleared by:

```
htmBullet arg;char;ct
⍝ Set bullet char for indented paras and optionally start counter
⍝ Allows strings such as '#.' to insert numeric counter
⍝ and '(@)' to count (a) this (b) that.
⍝ Note the kludge to deal with << htmBullet '@' 12 >>

 :if 2>|≡arg          ⍝ Simple
   :if 807=⎕DR arg    ⍝ Heterogeneous
     arg←(⊂,¯1↑arg),¯1↑arg
    :else
     arg←⊂arg
   :end
 :end
 (char ct)←2↑arg,1
 htm_bullet←⍪,char ⋄ htm_bulletct←ct
```

That really is all there is to it! Here is the test function again with some lines added to show the multi-level indenting in action:

```
⍝ Now some more detailed points ...
 htmPlace 'Key points to note are:'
 htmIndent 36 ⋄ htmBullet '#' ⍝ Counter required
 htmFlow 'Firstly it sells lots' 'Secondly it is superb value'
 htmIndent 48 ⋄ htmBullet '⍪' ⍝ Simple bullets
 htmFlow 'cheap to buy' 'low cost of ownership'
 htmIndent 36 ⍝ back down a level
 htmFlow 'Thirdly it is widely ... ftware stores worldwide.'
 htmIndent 0
 htmFlow 'So now you know.'
```

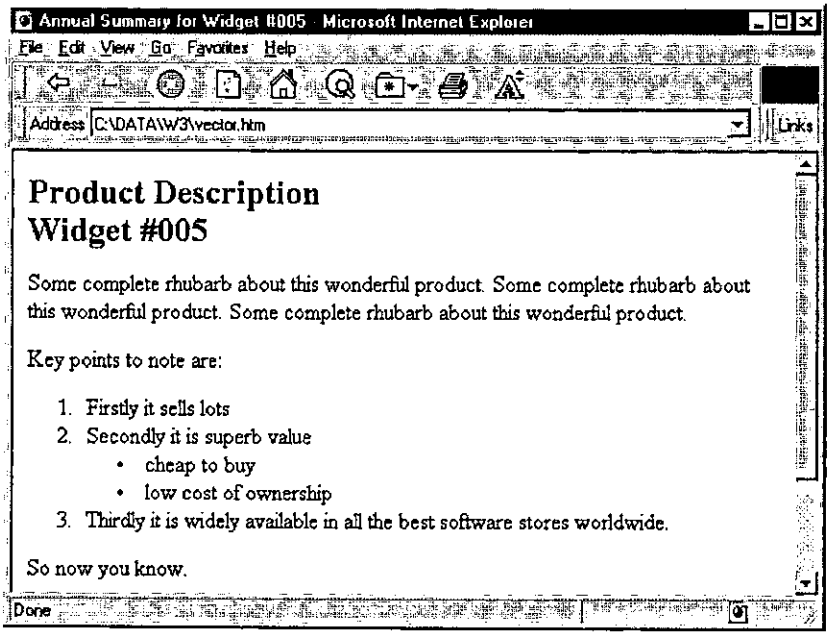
There is a particularly unpleasant kludge in the function to get around the possibility that someone may request `htmBullet '@' 12` which looks for all the world like a nested array, but is actually a two element *simple* vector. The more I encounter this sort of thing, the more I suspect that we must go back to VS APL and start all over again with a decent implementation of boxed arrays!

The equivalent Dyalog code fragment is ...

```
:If 2>|≡arg          ⍝ Simple
  :If 326=⎕DR arg    ⍝ Heterogeneous
```

... which is interesting in that here the result from ⎕DR does not distinguish the heterogeneous from the nested case.

... and here is the final result:



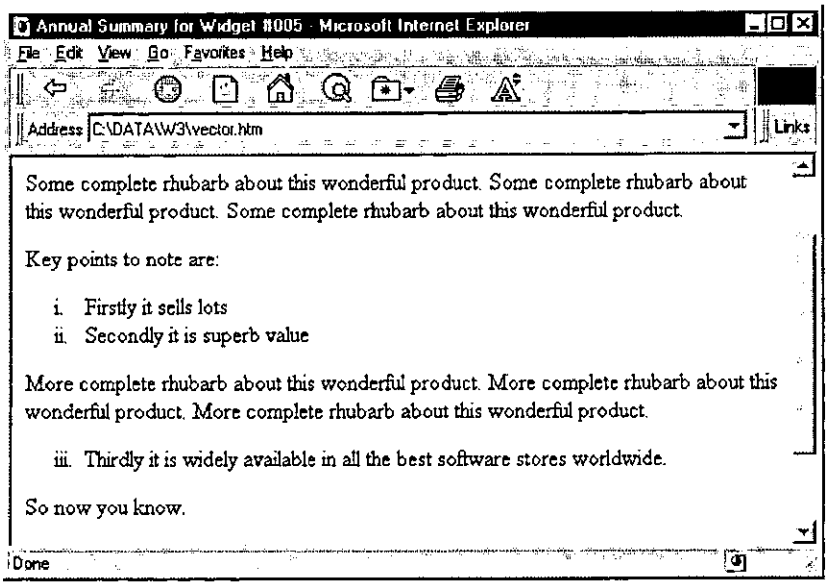
Some Other Settings

As you can see from the code, this will also accept `htmBullet '@.'` to set an alphabetic sequence, and `htmBullet '(∧)'` for roman numbering. You can preset the counter with an additional parameter, such as `htmBullet '@' 3`, for example to resume an interrupted list:

```
htmIndent 36 ◊ htmBullet '(∧)' ⍝ Roman counter
htmFlow 'Firstly it sells lots' 'Secondly it is superb value'
htmIndent 0
htmFlow ⍷3⍴⊂'More complete rhubarb about this wonderful product. '
htmIndent 36 ◊ htmBullet '(∧)' 3 ⍝ Roman counter from 3
htmFlow 'Thirdly it is widely ... ftware stores worldwide.'
```

... although the Roman sequence looks pretty strange without the enclosing parentheses.

Anyway, it works as well as could be expected ...



Coming Soon

The next article in the set will cover simple data-entry forms, in particular it will show you how to convert from a standard Causeway/AP124 form definition to something that shows up nicely in your browser, preserving as much of the layout as is reasonable. If I get around to it, it will also have the code to parse the resulting ASCII string which arrives over email when some hopeful user fills in your form and hits <Submit> to mail the request to you.

References

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

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

[3] *HTML Basics for APL+Win*, Vector 13.4 page 84

Using APL and J in Conjunction to Improve System Validity

by Donald B. Pittenger (dbpitt@msn.com)

Programming languages are usually seen in terms of opposition or contrast to one another: FORTRAN vs. Pascal, COBOL vs. PL/I, APL vs. J. Yes, definitely APL versus J. *Vector* and the Internet comp.lang.apl site are peppered with comparisons and contrasts as well as outright brickbats regarding the two languages (or is it dialects — I hope this paragraph does not start yet another holy war before I have even gotten to my main point).

I contend that there can be situations where APL and J can be used in a cooperative manner where the *goal is maximising software system validity.* This article is intended to explain my case.

Validating Software Systems

Give me some slack here: I'm a demographer, not a trained computer scientist. But I do write a lot of software to generate data to sell to clients, and I get bitten by software errors more often than I like. No doubt many of you are in a similar situation.

What is a 'valid' software system? For this article, I define it as one that does what the system designer intended. To take a simple case, a valid system would have no array indexing errors. I remember an instance where I got males and females reversed in one equation — just a matter of typing a 1 instead of a 2 by mistake. The system did not crash, so there was no obvious sign of trouble. The output looked a little odd, but nothing clearly unreasonable. I discovered the error months later when going over the code for some reason.

Is there any way to guarantee system validity? I say no, though there might be a few who disagree. About the best one can do is implement a practical validation procedure which reasonable observers agree is likely to locate a good many errors. If the system has any size or complexity, errors are almost certain to occur no matter what precautions are taken. Eventually, some of these will reveal themselves to users of the system or its output. Other errors (presumably small ones) might never be discovered.

Validation procedures can include code review, graphing output, and flagging results that might be considered questionable. There are others. But, in my opinion, *there is only one best way*. Read on.

The Solution is (Gasp!) Writing It Twice

Please calm down. I think I am quite reasonable when I assert that *the best way to validate software systems is to write two systems that use the same input and compare the output*. Admittedly, this is not always practical. The balance of this article sketches ways this can be done, given certain conditions. In brief, I propose that code be written in both APL and J, and I further claim that it is the use of these languages that makes the task practical at all. (Okay, I called them separate languages again. Please bear with me, because I will do so for the balance of the article.)

Where did I get the notion that writing a system twice is a practical proposition? Well, if you read the last paragraph of my article about the SAS IML matrix language in the July, 1997 *Vector*, you will note that I mentioned writing a system in J as well as IML. Because of my unfamiliarity with IML and SAS, as well as the lack of anyone to give me advice on how to build IML systems, my strategy was to restrict the IML system to core number-crunching chores. A good deal of input (rates, parameters, etc.) was computed off-line under J, but could have been done using a spreadsheet. Thus, the core system was small — about a dozen pages of code in small type. Partly because I didn't trust my IML skills and partly because I wanted a backup system in a language I knew, I wrote a J system that used the same input files as the IML system and which produced the same format of output files. (The files are *.txt type — the file you get when you save an Excel spreadsheet as text: basic ASCII with tab delimiters.)

One result of my efforts was that it was now possible to compare output of the two systems number-by-number. So I did compare numbers, and where there were differences, I checked both systems to try to figure out which one was in error. There must have been six or eight cases where differences were found, and the errors were fairly evenly balanced between IML and J. As I write this, the numbers agree, so I am confident that the system has been validated. Yes, there might be design mistakes, and there might be input errors, but the systems themselves are highly likely to be doing what they are supposed to be doing.

Where This Sort of Validation is Practical

Let me move from a specific case to discuss dual-language system validation in terms of APL and J, rather than J and IML.

I contend that J can be used to validate APL systems and APL can be used to validate J systems. It is also possible to validate APL by APL and J by J, but this assumes different programmers for each version. Where there is only one programmer, two languages should be used to limit the possibility of code being subconsciously repeated. Also, a one-language, one-programmer rewrite might easily include the use of common utility code, which might itself be flawed. (The IML/J exercise mentioned above revealed one error in my J utility code toolkit.)

To me, APL and J are well suited to dual-programming. They have enough similarities that a programmer of one language does not have to experience a paradigm shift to learn the other. But the differences are great enough that the programmer has to think in different terms to code the same process in each language. I have been programming in APL since 1983, and have become so proficient that a good many of my coding errors are the result of writing too quickly — slapdash stuff. I am far more cautious when writing J. I've been using it fairly heavily for about a year, and have avoided hooks, forks, etc., because (1) I have yet to confront a situation where the potential gain seems to exceed the pain of learning them, and (2) I find that even basic, APL-like J is not easy. I have a general sense of frames and function rank, yet nevertheless find myself constantly testing code segments on simple arrays just to be sure I've gotten things right. And zero-origin counting is not what I grew up with, so indexing always takes thought and maybe even doodling a few numbers or diagrams on a scratchpad. Indeed, my mind is working quite differently when programming in one language or the other, and this is exactly what is needed for one-programmer system validation.

By the way, I ought to make it clear that J code should not include tacit functions: the APL and J ought to be in the form of conventional programs whose listings can be compared and the behaviour of each variable traced. In tacit programming, the variable 'disappears' so far as the code is concerned. And you can't test the data flow of a tacit expression directly, if I understand the concept correctly.

One of the most important reasons why I think APL and J are good dual-programming tools is coding speed. APLers tend to throw a lot of speed factors around, especially lines of code compared to doing the same job in a compiled language. My impression is that the range runs from three to five times coding speed improvement. To use numbers loosely, assume APL and J can be written equally quickly by a programmer experienced in both. And take the more conservative end of the speed range. This implies that, compared to writing in a compiled language, the same system can be written in *both* APL and J in *two-thirds the time*. Or 40 per cent of the time, if the high-end speed advantage is

accepted. This, coupled *with the payoff of less time spent in the future tracking down system bugs,* means that the array-language programmer still retains the development-time advantage over competitors using conventional languages!

Potential Problem Areas

Nothing in this world is perfect. There are several practical matters standing in the way of dual-language validation in APL and J. These need to be discussed, lest we get carried away by the prospect of theoretical gains.

Before dealing with problem areas, let us start by describing an ideal dual-language validity testing setting because, by showing an ideal, we can reveal what has been abstracted out of practical circumstances. Actually, the IML/J example mentioned above came pretty close; let's use a polished version of it. Here are salient features:

1. The core computational task is well-structured. There can be alternative subroutines, but most of the variation is in the input data and parameter values. Essentially, the core is a batch-process wrapped in some sort of interface.
2. The identical input must be brought to each language system test run so that any differences in output must be due to system flaws.
3. The simpler the user interface (UI) the better — at least for testing purposes. This is an enabling factor relating to the need to have each system running identical input. Since UI's deal with input selection and flow of control, steps must be taken to assure common system control. The ideal setting would have no interface in the generally accepted sense; it would be a software 'breadboard' or workbench where consistent inputs are introduced as globals, passed data, or whatever is the design of the production system. In other words, for testing, all the data/parameter selecting normally done via the UI can be assumed to have occurred, and the core routines are now about to begin their work. For complex systems, modules might have to be isolated in this fashion for individual testing — particularly if the complete system requires UI actions to bridge module operations.
4. Output data should be in the form of tables containing summary data, descriptive data such as rates or shares, and blocks of raw numbers. Summary and rate data are useful for spotting the general locations of errors, detailed data confirm that error fixes work at the lowest level. Demographic data I have used for testing contained single-year-age data as well as age groups, population totals, and totals for process outcomes such as births, deaths, and migrants.

From the above, there are some obvious and not-so-obvious practical problem areas. But the key item is how well the test systems can be isolated from UI and

language environment peculiarities. Make every reasonable effort to create test-beds. Otherwise, you are likely to encounter the following problems:

One obvious problem is the UI. My experience does not extend to mainframe or minicomputer APL, but I know from experience differences between APL*PLUS and APL2/PC under MS-DOS, not to mention APL+WIN, Dyalog APL/W, ISIAPL, and J under Windows. The UI, whether it is basically a fancy menu or something far more flexible, can influence much data selection and program flow. The dual-language problem is how to get the same input into these separate code blocs. If the UI was via Delphi or Visual Basic, and the J and APL were DLLs chewing on data furnished by the UI system, there would be little difficulty. (Few APLs can act as DLL servers as this is written, but in future most will.) Problems can occur under Windows if you use a language vendor-based UI. For example, what if the target system was written in Dyalog APL/W and used the grid object? J under Windows does not yet have such a thing. A problem-simplifying Windows UI solution (though not for grid objects) would be Causeway — except that there is not yet a Causeway for J.

Another problem is data input — in particular, what can be done with target systems that make use of APL component files? J does not read APL component files. The same applies to data stored as part of an APL workspace. As a practical matter, dual-language validation systems are best suited to settings where data flow to and from 'standard' file types, be they *.txt or files read from data bases using 'standard' linkage software.

I know this is easy to say but hard to do in the real world, but: if I were developing *new systems* in APL and hoped to perform dual-language validation using J, I would try to design the production system (not just a breadboard system) so as to eliminate all aspects of APL data storage as it was classically practised. This includes the workspace as well as component files. Okay, you can't run APL without a workspace. But, aside from loading-speed considerations, why should APL systems be *stored* as workspaces? The fear of destroying programs by mistake led me to abandon workspace-based systems soon after I started using APL. I store programs as individual files grouped within directories. It is a flexible and safe solution. It also helps facilitate dual-language validation because specific program sets can be assembled fairly easily for testing as functional blocs as noted above. If APL is to have a future, dysfunctional baggage must be ruthlessly abandoned where necessary.

Conclusion

Validating software systems by writing them in two different languages can be a practical matter if those languages are APL and J, provided that identical input can be brought to each comparative test. The key element in making this proposal practical is the programming speed of the two languages — writing a system in both APL and J to assure validity can be done faster than programming it once (without attempting validation) using a conventional language.

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

Hackers' Corner: Dyalog APL and the Windows 95 Registry

by John Sullivan (john@yddraiggoch.demon.co.uk)

Why Use the Windows Registry?

Windows 3.*x* used initialization files, which had the extension .ini, to hold the settings for Windows itself and for applications written to run under Windows. There were two types of initialization file, *system* and *private*, although in many cases programmers were lazy and wrote information that rightly belonged in a private initialization file into a system file, such as system.ini or win.ini. Since the largest size .ini file that Windows 3.*x* could handle was 64k, this caused problems whenever users installed "too many" of these applications. On the other hand, some programs used private initialization files to store system information; for example Microsoft Publisher stored its font information in a private file. Make the font unavailable and the program crashes!

With Windows 95 the Registry was introduced in order to keep all initialization data together, rather than in a succession of fragmented .ini files. Although support for .ini files is still maintained (and has even been enhanced) in Windows 95, because, obviously, programs written for Windows 3.*x* will require access to .ini files when run under Windows 95, performance when accessing the Windows registry is significantly faster than when accessing .ini files in almost all circumstances. There are no restrictions to the size of the registry, whereas the functions that write to .ini files fail when the file approaches its size limit of 64k.

Since Windows 95 applications are supposed to write to the registry rather than to initialization files, developers who fail to use the registry can produce applications that are not completely compliant with the Win95 software design guidelines.

The Functions

There are six of these:

Open, which opens a path in the registry and returns a handle
GetInt, which gets an integer value from the registry
GetString, which gets a character string from the registry
PutInt, which puts an integer on the registry
PutString, which puts a character string on the registry
Close, which closes the registry when you've finished.

Dyadic Systems have already provided some functions to perform registry operations: these are in the Workspace Documentation namespace in the session. We can reduce our workload by making use of them, giving them new names and putting them in our own namespace for completeness. I put these functions in a namespace called *Reg* to keep them together and out of the way. The functions we reuse (thanks to Dyadic for permission to list these) are:

```
Open←⎕SE.WSDoc.GetRegKeyHandle
GetInt←⎕SE.WSDoc.GetRegKeyValue
PutInt←⎕SE.WSDoc.PutRegKeyValue
```

```
     ∇ HANDLE←GetRegKeyHandle KEY;CURRENT_USER;ALL;RegCreateKeyExA
[1]    ⍝ Get a handle to a Registry key as a subkey of HKEY_CURRENT_USER
[2]    ⍝ The key is created if it doesn't already exist
[3]
[4]    CURRENT_USER←2147483649          ⍝ HEX 0x80000001
[5]    ALL←2035711                      ⍝ HEX 0x1F0FFF
[6]
[7]    ⎕NA'I ADVAPI32.dll.C32|RegCreateKeyExA U <0T I <0T I I I >U >U'
[8]    HANDLE←⊃2⊃RegCreateKeyExA CURRENT_USER KEY 0 '' 0 ALL 0 0 0
     ∇

     ∇ VALUE←KEY GetRegKeyValue NV;SUBKEY;RegQueryValueExA;RC;VAL
[1]    ⍝ Gets the value of a Registry SUBKEY or default DEFAULT
[2]    ⍝ KEY is the handle for an existing Registry Key
[3]    SUBKEY VALUE←NV
[4]    ⎕NA'I ADVAPI32.dll.C32|RegQueryValueExA U <0T I I >I4 =I4'
[5]    RC VAL←2↑RegQueryValueExA KEY SUBKEY 0 0 0 4
[6]    →(RC≠0)/0
[7]    VALUE←VAL
     ∇
```

```
     ∇ HANDLE PutRegKeyValue NV;SUBKEY;VALUE;RegSetValueExA
[1]   ⍝ Stores the value of a Registry SUBKEY
[2]    SUBKEY VALUE←NV
[3]    ⎕NA'I ADVAPI32.dll.C32|RegSetValueExA U <0T I I <I4 I4'
[4]    VALUE←RegSetValueExA HANDLE SUBKEY 0 4 VALUE 4
     ∇
```

The syntax of these functions is as follows, where key is a subkey of HKEY_CURRENT_USER, and that part of the key does not need to be given.

```
handle←Reg.Open key
value←handle Reg.GetInt subkey default
handle Reg.PutInt subkey value
```

It is important to close all open registry handles as soon as you've finished with them, in order to maintain data integrity (what do you do if your system crashes between updating the registry itself and the time when Windows actually gets around to writing the updates to disk?) and also to free up Windows resources.

The other functions are easily written, copying the style of the Dyadic-supplied functions:

```
     ∇ VALUE←KEY GetString NV;LEN;SUBKEY;DEFAULT;RegQueryValueExA;RC;VAL
[1]   ⍝ Gets the value of a Registry SUBKEY or default DEFAULT
[2]   ⍝ Key is the handle for an existing Registry Key
[3]   ⍝ Function by JS based on ⎕SE.WSDoc.GetRegKeyValue
[4]    SUBKEY VALUE←NV
[5]    ⎕NA'I ADVAPI32.dll.C32|RegQueryValueExA U <0T I I =0T =I4'
[6]    RC VAL LEN←RegQueryValueExA KEY SUBKEY 0 0(256⍴' ')256
[7]    :If RC=234          ⍝ More data (i.e. buffer is not big enough)
           RC VAL LEN←RegQueryValueExA KEY SUBKEY 0 0(LEN⍴' ')LEN
[8]    :EndIf
[9]    →(RC≠0)/0
[10]   VALUE←VAL
     ∇

     ∇ {RC}←HANDLE PutString NV;SUBKEY;VALUE;RegSetValueExA
[1]   ⍝ Stores the value of a Registry SUBKEY
[2]   ⍝ HANDLE is the handle for an existing Registry Key
[3]   ⍝ Function by JS based on ⎕SE.WSDoc.GetRegKeyValue
[4]    SUBKEY VALUE←NV
[5]    ⎕NA'I ADVAPI32.dll.C32|RegSetValueExA U <0T I I <0T I4'
[6]    RC←RegSetValueExA HANDLE SUBKEY 0 1 VALUE(1+⍴VALUE)
     ∇
```

```
      ∇ {RC}←Close HANDLE;RegCloseKey
[1]    ⍝ Close an open registry key
[2]     ⎕NA'I ADVAPI32.dll.C32|RegCloseKey U'
[3]     RC←RegCloseKey HANDLE
      ∇
```

Example

If you want to change the value of your application's printer top margin to 36 points, saving the old value, you would key something like the following:

```
handle←Reg.Open 'Software\APLapps\application\Print'
old←handle Reg.GetInt 'TopMargin' 0
handle Reg.PutInt 'TopMargin' 36
Reg.Close handle
```

Other Considerations

If you try to open a non-existent subkey it (and all non-existent intermediate subkeys) will be created. This is a function of the registry interface. Similarly, you do not need to create a bottom-level container for your data: in the example above, if `TopMargin` did not already exist, the default value of 0 would be put into `old`, and the result of the `PutInt` call would be to create `TopMargin` and store its value.

Other registry functions are available, for example functions to delete entries. I have not coded those because up to now I have not needed them, being content to use the registry editor to delete unwanted entries, but those who do need them should have no trouble coding them yourself. (You would need deletion functions for an uninstall program, for instance.)

References

Any good book on the Windows 95 registry, or the *Microsoft Developer Network* CD-ROMs.

Technical Correspondence

From: David Piper 5th March 1998

After many years doing APL, followed by fewer years of retained interest in APL, I am still amazed by the abysmal quality of some of the code being published in Vector. One of the major objections to APL in the commercial world is the production of write-only code. The article in Vector 14.2 on Niven Numbers is a perfect exemplar of the coding style that creates such negative reactions. Whilst this article is academic in nature, Vector is a (arguably the) showcase for APL code. All code published here should be of a high quality, or at least ultimately form a contribution to high quality code (maybe as part of a learning exercise). Take the function *NIVEN2*; it can be criticized for at least the following *features*:

- use of single character names;
- multiple assignments on a single line;
- complex path analysis (75% of the lines include conditional branches).

The attempt to make the code more efficient (*NIVENB*) simply exaggerates these features and gives rise to an unrewarding 4% benefit. Clearly the structure of the code is not the source of the performance issue, it is far more likely that the algorithm is to blame. Algorithmically the functions given can be criticized on (at least) two fronts:

- repeated catenation of new result values to the result vector;
- evaluation of only one candidate result at a time.

Below is a version of the algorithm which avoids the repeated catenation issue. This is an easy change to make since we know from the right argument how many result values are required:

```
∇Z←NIVEND W;candidate;digits;pos
 Z←((⌈W÷10)⍴0)
 candidate←0
 pos←1
 →(W≤0)/end
next:candidate←candidate+1
 digits←((1+⌊10⍟candidate)⍴10)⊤candidate
 →(0≠(+/digits)|candidate)/next
 Z[pos]←candidate
 pos←pos+1
 →(W≥pos)/again
end:
∇
```

In performance terms this version shows a significant improvement (here are some relative timings for cases > 1000):

Function	1000	10000	100000
NIVEN1	1.07	20.85	755.55
NIVEND	1.14	15.92	204.33
Ratio	1.07	0.76	0.27

The penalty in performance below 1000 is certainly due to the "style" of the code - this algorithm implemented in the *style* of *NIVEN1* shows a constant improvement of about 10% over the more readable version presented above. For the 100000 case, the saving due to changing the algorithm is 73%, the saving from changing the code style is about 3%.

This illustrates a vitally important point:

> *as the amount of useful computation increases as a proportion of the total the overheads due to "style" or other factors become less important; in general pursuit of algorithm is more beneficial than sacrificing readability for marginal gains in efficiency.*

Despite the effectiveness of the gains made so far, greater gains become available by implementing the algorithm in a way suited to APL. The solutions presented are *scalar* in nature - each scalar is tested in turn. APL is an array processing language, so an *array* based solution should be sought. In contrast to the statement made in the original article that *...it would be a challenge to find an algorithm...* the changes required are fundamentally simple:

```
∇Z←A NIVENE W;candidates;digits;hits;max
 Z←((W⌈0)⍴0)
 candidates←⍳A
 →(W≤0)/end
next:max←(⍴⍴candidates)⊃candidates
 digits←((1+⌊10⍟max)⍴10)⊤candidates
 hits←0=(+⌿digits)|candidates
 Z←Z,hits/candidates
 candidates←candidates+A
→(W>⍴Z)/again
 Z←W↑Z
end:
∇
```

The algorithm implemented in this function generates groups of candidates and tests the content of each group in parallel. The size of the group is determined by the left argument to the function. At first sight this algorithm may seem inefficient since it potentially generates *too many* result values, however the cost of this marginal inefficiency is far outweighed by the gains resulting from the parallelism of the algorithm as a whole. The performance of this improved algorithm is illustrated in the following table:

Function	1000	10000	100000
NIVEN1	1.07	20.85	755.55
NIVENE	0.06	0.97	13.91
Ratio	0.06	0.05	0.02

The performance of the algorithm does depend on the value chosen for chunking size. An optimal choice seems to be a value close to that in the right argument, though the reasons for this are not clear. Re-writing this algorithm in the style of *NIVEN1* did not give a noticeable performance benefit (<1% in all cases compared to *NIVENE*). The function given can be subjected to a criticism made of the original - the use of repeated catenation to form the result. Note, however, that the catenation will be performed far fewer times and a (potentially large) number of results is appended in each operation.

The point is best made by the principle expressed in Normal Thomson's article in the same edition of Vector - *think, reason, express*. Once a simple method has been *thought* up, *reason* it through to create an algorithm suitable for array processing. Then this can be *expressed* in a clear and meaningful manner. In my experience, only rarely is there a need to sacrifice clarity for performance.

David Piper
11 Lingfield Road
Evesham
WR11 6XG

Email: DavidP@SelectST.com

An ODBC Browser Using SQL and Dyalog APL

by Richard Smith (richard@apl-385.demon.co.uk)

Introduction

Having not gone to Morten Kromberg's workshop on ODBC at APL96, I decided that I would have a go at it in Toronto. As I was working through the workshop, I realised I was having to type in things like `SQAInit`, `SQADo` and `SQAClose` a lot of times. So I made a basic Windows form with a field and a "Go" button, which ran `SQADo` with whatever was in the field, and "Connect" and "Disconnect" buttons which ran `SQAConnect` and `SQAClose`; this meant I didn't have to type it in each time.

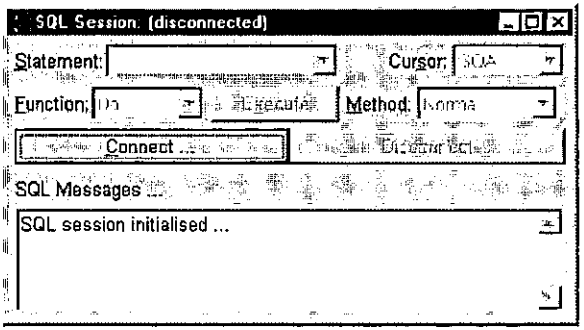
As I continued through the workshop, it became more and more complicated. As each new function was introduced by itself, naturally I added new controls to the form for each one. This meant that my form had some fairly unorthodox behaviour – the dropdown on the left, defining which function to perform, controlled the items shown in the right-hand dropdown; both together controlled the actions of the "Execute" button. This led to a rather large 'switch-case' statement in working out what to do.

Finishing it off

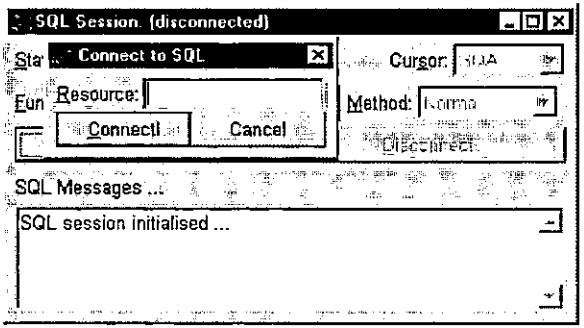
Annoyingly, near the end of the workshop my Dyalog interpreter crashed (the only time during the whole conference: what a surprise, just as I had a good program). However, this gave me the opportunity to rewrite the interface rather more sensibly. Using my father's portable, I wrote a dummy version (he didn't have ODBC installed), which let me play with graphic properties. Towards the end of the conference, I went back into the workshop room to write in the connections with Morten's functions and to test it. At this stage, it was a working product and did not have some of the 'fancy features' of the one now, but it was taking on the look of it.

During the Christmas holidays, my father, Adrian, thought it would be a good idea if it had a dropedit Combo for the statements, as in web browsers such as Netscape and Internet Explorer. Having done this, I was then told that it would be better as a multi-line field, to allow longer statements; however, I wasn't going to write that: come on Dyadic, let's have "Multi-Line Combo Boxes"!

The Final Design

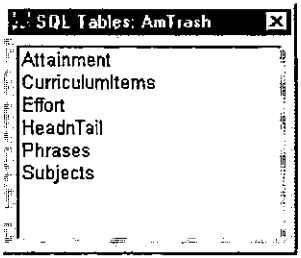


After fiddling with the *attach* property and other such things, I came up with this as my main form. The icon is an interesting feature: it's different every time! I make it using the APL expression `¯1+?32 32⍴16`, producing a random matrix of 16 colours. A useful point about the program is the way the focus moves at different times. At the start, it is on the "Connect" button. When this button is pressed, it brings up this window:



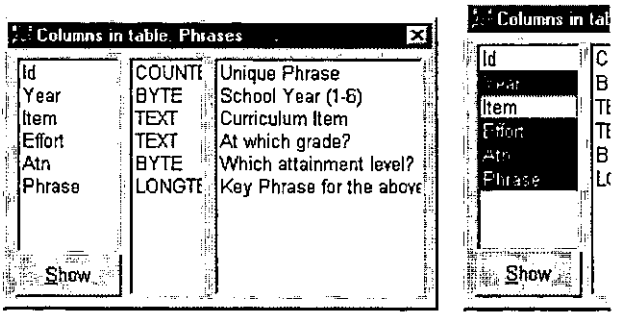
The focus is placed in the "Resource" field. Type in the name of the resource and press "Enter" or click "Connect!" to connect to the resource. If it connects successfully, the fields will be activated and the Tables window will be displayed.

This window shows the names of all the tables in the resource selected - here there are six - and the window can be resized to show any very long names.



By double-clicking on any of the table names, you can bring up the Columns window (see below). You can have as many of these windows open at one time as you like; double-clicking on a different table will bring up another Columns window. Double-clicking a name for which there is already a Columns window active will give the focus to that window.

Pressing "Enter" or clicking on "Show" will give the Return window for the table:

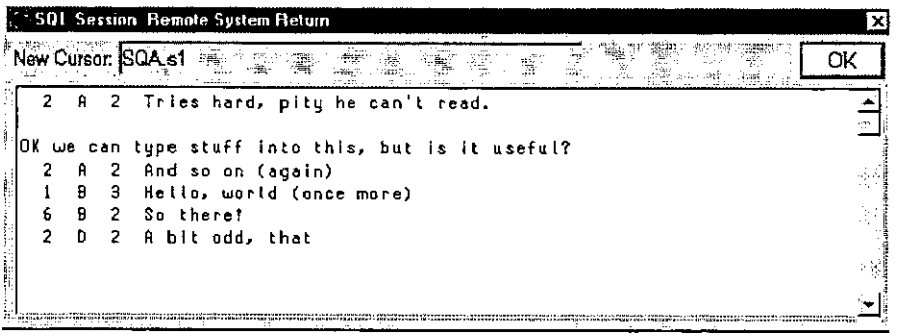


By selecting one or more column names and then pressing "Show", the Return window is shown for just those columns of the table.

This is done by the function `CreateQuery`:

```
    tab CreateQuery fields
[1] ⍝ Create an SQL call that shows the values of fields
[2] ⍝ <fields> in table <tab>.
[3] :If 0∈⍴fields
[4]   ('Do' 'Normal')Run('Select * from ',tab)
[5]   :Else
[6]   ('Do' 'Normal')Run('Select ',(¯1↓∊'"',¨fields,¨⊂'",'),' from ',tab)
[7] :End
```

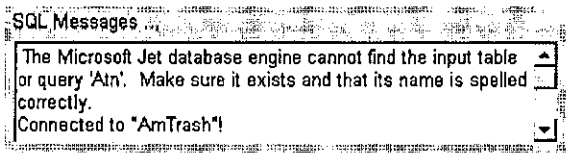
Line 3 checks to see if any fields have been selected. If not, it runs line 4, which selects all the columns from the selected table. Otherwise, it runs line 5, which shows just the columns given to the function in the variable `fields` (Year, Effort, Atn and Phrase in this example). `Run` puts the query in the list of statements and displays the result in the Return window:



The display here is a little confusing. The first column, Year, is 2 2 1 6 2; the second, Effort, is A A B B D; the third, Effort is 2 2 3 2 2. The text "OK we can type stuff into this, but is it useful?" is part of the Phrase column of the *first* entry.

Extras

This mouse interface can't do every database action: for instance, it can't do cross-table queries. However, to do another action via SQL, type in the statement you want to run in the "Statement" field on the main form, then press "Enter" or click on "Execute" to run it. If you have typed it in correctly, the output will appear in the Result window; if not, the error message will appear in the "Messages" field on the main form:



To re-use a statement from earlier (including those made using the "Show" button on the Columns window), click on the drop-down button by the "Statement" field to bring down the Combo box:

This shows all the statements used since the program was started, the most recent at the top. If you re-use a statement, it will riffle to the top of this list. When a statement is selected, use the "Execute" button to run it.

Richard Smith
Brook House
Gilling East, York

Note: The article and the workspace are available at my webpage at:

www.apl-385.demon.co.uk/rcs/odbc.htm

Object Oriented Programming with APL+Win (Part II)

by Eric Lescasse (eric@lescasse.com)

I want to thank Chris Lee from Softmed who inspired me with his UDC (User Defined Classes) concept.

Introduction

This long article, published in Vector in 2 parts, is extracted from the Monthly **APL+Win Training Program** available on the Internet from the **Lescasse Consulting** Web site at `www.lescasse.com`. It represents one of 3 chapters included in the APL+Win Training for May 97.

Inheritance

The way of programming with `Qwi` described in this Chapter is really **Object Oriented Programming with APL+Win**. To convince you about this assertion, read on.

First, we are really creating new classes of objects. When you create a new class, you choose a class name for it (example: **Form** or **Form5** or **MaskEdit**, etc.) and then you create one APL function with a name that matches the new object's class name you've chosen.

Remember that you can replace existing APL+Win classes (like **Form**) with your own class of the same name.

You may also overload existing class properties or methods with your own behaviour. This is exactly what we have done in all examples so far with the **New** method, which does exist for the APL+Win standard **Form** class, but which we overloaded with additional functionality in our **Form1** to **Form5** classes.

Second, we can easily go one step further and use **inheritance**, one of the basic concepts of all Object Oriented development systems. We will now describe how to do that.

Let's suppose that we would like our new **Form** class to behave exactly like the **Form5** class except that the **Esc** key would be a shortcut key used to close the form. We could change our `Form5` function by adding one more line in the

overloaded **New** method code and rename *Form5* to *Form6* (which you will find in workspace *QWI.W3*). The line to add would be:

```
[24]  R←(A,'.bnEsc')⎕wi'New' 'Button'('size' 0 0)('style' 2)  ⍝ Esc button
```

This line creates an invisible button (invisible because its size is set to 0 0!) with style 2 on our form. Because this button has style 2, it is known by the APL+Win system as a **Cancel** button which is automatically clicked by pressing the **Esc** key, therefore closing the form.

So, we have perfectly solved our problem of creating a new class of forms which would react to pressing the **Esc** key by closing themselves. However, there are several drawbacks to this approach:

- first, we are duplicating the *Form5* code to become *Form6* and losing precious workspace room
- second, and more importantly, if we need to make any change in our **Form5** class, we must make the same changes in **Form6** class, so that **Form6** objects continue to behave exactly like **Form5** objects (except for the added Esc key functionality)

Could we try to take advantage of the Object Oriented **inheritance** concept instead? Yes, and it is ***very*** easy. Let's just design a new **Form7** class as follows:

```
    ∇ A Form7 B;R;class
[1]   ⍝∇ A Form7 B -- User Defined Form7 Class
[2]   ⍝∇ A ↔ object name
[3]   ⍝∇ B ↔ 'property'
[4]   ⍝∇    or 'property' value
[5]   ⍝∇    or 'Method'
[6]   ⍝∇    or 'Method' argument1 ... argumentN
[7]   ⍝∇ Requires: (F) QwiRegister
[8]   ⍝∇ Copyright (c) 1997 Eric Lescasse [9mar97]
[9]
[10]  :select ⍎B
[11]  :case 'New'                    ⍝ constructor
[12]      A Form5 B                  ⍝ inherits Form5 class
[13]      QwiRegister A              ⍝ save class name
[14]      R←(A,'.bnEsc')⎕wi'New' 'Button'('size' 0 0)('style' 2)
[15]
[16]  :else
[17]      A Form5 B                  ⍝ inherits Form5 class
[18]
[19]  :end
    ∇
```

That's a very simple and crystal clear way of implementing inheritance.

When the *Form7* class function is called with an argument starting with *'New'* to create a new instance of a **Form7** kind of a form, the *Form5* **New** method code is used instead (**line 12**) and is simply followed by the additional behaviour we wanted (**line 14**). Note that we must not forget to register our new class (**line 13**).

For all other **Form7** class properties and methods, the behaviour is inherited from the **Form5** class (**line 17**).

You may want to check if this works OK. Try:

```
      'ff' Qwi2 'New' 'Form7'
ff
```

click in the form to set the focus to it and press the **Esc** key: this closes your form.

Then try:

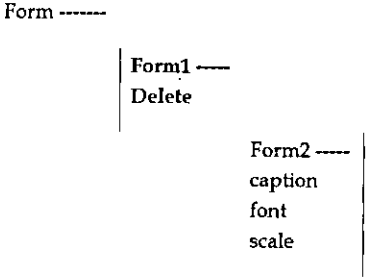
```
      'ff' Qwi2 'New' 'Form7' ('ontop' 1)
ff

      'ff' Qwi2 'Move' 'bottom' 'right'
192 256
```

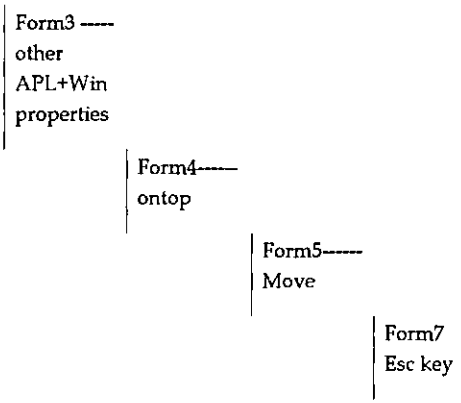
The form moves as expected. The **ontop** property and the **Move** method are purely inherited from the **Form5** class!

I suppose you see how this brings a terrific power to APL+Win programming!

APL+Win allows you to do **Object Oriented Programming** and in a manner which is most simple.

Having understood inheritance as shown above, we could have designed our **Form1** to **Form5** classes as a hierarchy of classes, all descendants from the APL+Win **Form** class, according to the following tree:





In the above tree, I have shown at each level of the tree, the changed or added properties and methods.

Exercise

I leave it as an exercise for you to start a new workspace, copy functions `Qwi`, `HANDLERFOR`, `Form1` from workspace `QWI.W3` in it and rewrite `Form2`, `Form3`, `Form4`, `Form5` and `Form7` implementing inheritance at all levels.

Then add a new property to the **New** method of the **Form1** class, like changing the default form `size` to **10 30**, and check that all other classes (**Form2** to **Form7**) do inherit correctly from the **Form1** class.

Polymorphism

Another unique characteristic of **Object Oriented Programming** is **polymorphism** whereby different objects respond to the same message with their own unique behaviour.

How can we use polymorphism with our new objects? The answer is simple: if you assume that invoking an object method is in fact sending a message to the object instructing it to run a named piece of code (the method), then we simply need to implement methods sharing the same name in different ways according to the classes in which they are implemented.

An example will help understand this polymorphism concept. Let's define 2 new classes of objects **Form8** and **Form9** which both inherit from class **Form7**. In both

Form8 and **Form9** classes, we will implement a *SetDefaultColor* method, which sets the form background colour to some default class color when it is invoked. However we will design *SetDefaultColor* so that it changes the colour of **Form8** forms to black and the colour of **Form9** forms to white. Here are our 2 new class functions:

```
    ∇ A Form8 B;R;class
[1]   A∇ A Form8 B -- User Defined Form8 Class
[2]   A∇ A ↔ object name
[3]   A∇ B ↔ 'property'
[4]   A∇    or 'property' value
[5]   A∇    or 'Method'
[6]   A∇    or 'Method' argument1 ... argumentN
[7]   A∇ Requires: (F) QwiRegister
[8]   A∇ Copyright (c) 1997 Eric Lescasse [9mar97]
[9]
[10]  :select ↑B
[11]  :case 'New'                  A constructor
[12]      A Form7 B                A inherits Form7 class
[13]      QwiRegister A            A save class name
[14]
[15]  :case 'SetDefaultColor'      A SetDefaultColor method
[16]      A ⎕wi 'color' 0          A set background color to black
[17]
[18]  :else
[19]      A Form7 B                A inherits Form7 class
[20]
[21]  :end
```

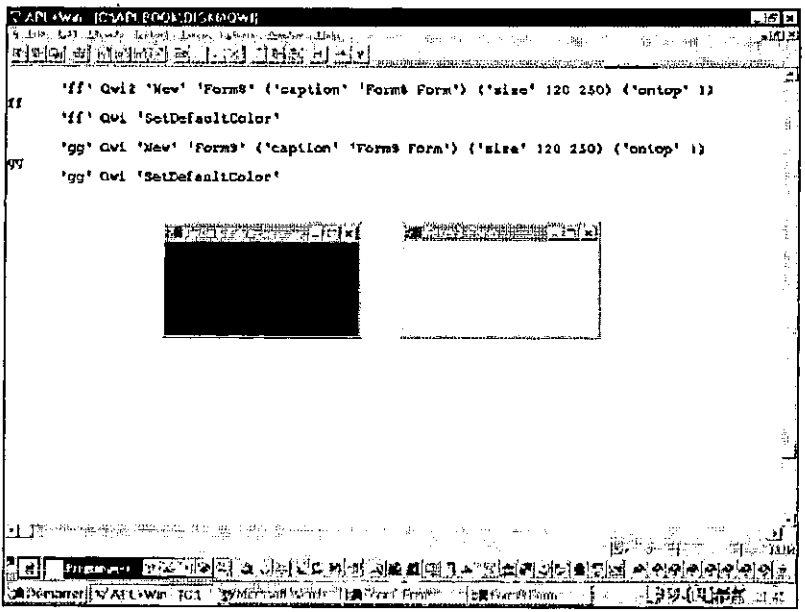
and:

```
    ∇ A Form9 B;R;class
[1]   A∇ A Form9 B -- User Defined Form9 Class
[2]   A∇ A ↔ object name
[3]   A∇ B ↔ 'property'
[4]   A∇    or 'property' value
[5]   A∇    or 'Method'
[6]   A∇    or 'Method' argument1 ... argumentN
[7]   A∇ Requires: (F) QwiRegister
[8]   A∇ Copyright (c) 1997 Eric Lescasse [9mar97]
[9]
[10]  :select ↑B
[11]  :case 'New'                  A constructor
[12]      A Form7 B                A inherits Form7 class
[13]      QwiRegister A            A save class name
[14]
[15]  :case 'SetDefaultColor'      A SetDefaultColor method
[16]      A ⎕wi 'color' (3⍴255)    A set background color to white
[17]
[18]  :else
[19]      A Form7 B                A inherits Form7 class
[20]
[21]  :end
    ∇
```

Now let's try to create instances of the **Form8** and **Form9** classes, which we will distinguish by changing their caption property and let's send them a *SetDefaultColor* message.

```
        'ff' Qwi2 'New' 'Form8' ('caption' 'Form8 Form') ('size' 120 250) ('ontop' 1)
ff
        'ff' Qwi2 'SetDefaultColor'


        'gg' Qwi2 'New' 'Form9' ('caption' 'Form9 Form') ('size' 120 250) ('ontop' 1)
gg
        'gg' Qwi2 'SetDefaultColor'
```

Here is what you will see.



This is exactly what **polymorphism** is.

Polymorphism refers to the ability of a given message to assume different functions, depending upon to what object it is sent.

Using Superclass Methods

It may sometimes happen that you would like to use not the class definition of a (possibly polymorphic) method, but its **superclass** definition instead. A **superclass** is simply the parent class which a given class inherits from.

For example, the **Form8** superclass is **Form7**.

Let's assume that we have also implemented a *SetDefaultColor* method in the *Form7* class. Change class function *Form7* as follows and rename it as *Form10*:

```
    ∇ A Form10 B;R;class
[1]   A∇ A Form10 B -- User Defined Form10 Class
[2]   A∇ A ↔ object name
[3]   A∇ B ↔ 'property'
[4]   A∇    or 'property' value
[5]   A∇    or 'Method'
[6]   A∇    or 'Method' argument1 ... argumentN
[7]   A∇ Requires: (F) QwiRegister
[8]   A∇ Copyright (c) 1997 Eric Lescasse [9mar97]
[9]
[10]  :select ↑B
[11]  :case 'New'                  ⍝ constructor
[12]      A Form5 B                ⍝ inherits Form5 class
[13]      QwiRegister A            ⍝ save class name
[14]      R←(A,'.bnEsc')⎕wi'New' 'Button'('size' 0 0)('style' 2)
[15]
[16]  :case 'SetDefaultColor'      ⍝ SetDefaultColor method
[17]      A ⎕wi 'color' 255        ⍝ set background color to red
[18]
[19]  :else
[20]      A Form5 B                ⍝ inherits Form5 class
[21]
[22]  :end
    ∇
```

Then change function *Form8* to *Form11* as follows:

```
    ∇ A Form11 B;R;class
[1]   A∇ A Form11 B -- User Defined Form11 Class
[2]   A∇ A ↔ object name
[3]   A∇ B ↔ 'property'
[4]   A∇    or 'property' value
[5]   A∇    or 'Method'
[6]   A∇    or 'Method' argument1 ... argumentN
[7]   A∇ Requires: (F) QwiRegister
[8]   A∇ Copyright (c) 1997 Eric Lescasse [9mar97]
[9]
[10]  :select ↑B
```

```
[11]  :case 'New'                  ⍝ constructor
[12]      A Form10 B               ⍝ inherits Form10 class
[13]      QwiRegister A            ⍝ save class name
[14]
[15]  :case 'SetDefaultColor'      ⍝ SetDefaultColor method
[16]      A ⎕wi 'color' 0          ⍝ set background color to black
[17]
[18]  :else
[19]      A Form10 B               ⍝ inherits Form10 class
[20]
[21]  :end
    ∇
```

We now have a new class of forms: **Form11** which knows about the `SetDefaultColor` message and inherits the **Form10** class.

The **Form10** class also understands the `SetDefaultColor` message, albeit in a different manner and the **Form10** class inherits the **Form5** class.

When we are working with a **Form11** kind of a form, sending a `SetDefaultColor` message to the form results in running the `SetDefaultColor` method of the **Form11** class.

If you want to run the `SetDefaultColor` method instead in the **Form11** superclass (i.e. in class **Form10**) you should send a **super SetDefaultColor** message to the **Form11** form.

Example:

```
      'ff' Qwi3 'New' 'Form11' ('caption' 'Form11 Form')('size' 120 250)('ontop' 1)
ff
```

Let us first invoke the **Form11** class `SetDefaultColor` method:

```
      'ff' Qwi3 'SetDefaultColor'
```

This results in a **black** form being displayed on the screen.

And now, let's call the **Form11 superclass** `SetDefaultColor` instead:

```
      'ff' Qwi3 'super SetDefaultColor'
```

This results in a **red** form being displayed on the screen.

For this to work properly, you need to change utility `Qwi2` to now be `Qwi3`, as follows:

```
    ∇ R←A Qwi3 B;C;D;E;F;G;I;P;T;⎕elx
[1]   A∇ R←A Qwi3 B -- ⎕WI emulator
[2]   A∇ ⎕wself←'object_name'  Qwi3 'New' 'class_name'
[3]   A∇ ⎕wself←'object_name'  Qwi3 'New' 'class_name'('prop1' value1)...
[4]   A∇ res←  ['object_name'] Qwi3 'prop'
[5]   A∇       ['object_name'] Qwi3 'prop' value
[6]   A∇       ['object_name'] Qwi3 ('prop1' value1)...('propN' valueN)
[7]   A∇ res←  ['object_name'] Qwi3 'Get_Method' [arguments]
[8]   A∇       ['object_name'] Qwi3 'Set_Method' [arguments]
[9]   A∇ Copyright (c) 1997 Eric Lescasse [8mar97]
[10]
[11]  ⎕elx←'⍎HANDLERFOR ⎕DM'
[12]  :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end     A ⎕WSELF if no left arg
[13]  :if 1=≡B ⋄ B←,⊂B ⋄ :end            A be sure B is nested
[14]  :if 'New'≡↑B                       A if method is 'New'
[15]      C←2⊃B                          A get specified class name
[16]      B←(⊂'New' C),2↓B               A rebuild argument
[17]  :elseif 0≠⍴A ⎕wi'self'             A else if object A exists
[18]      :if 0≡C←↑D←A ⎕wi'⍙class'              A get registered class name
[19]          ⎕error'Use ⎕WI instead of Qwi3'   A if none signal error
[20]      :end
[21]  :else                                     A if not 'New' & not an object
[22]      ⎕error'Unknown object: ',A            A signal error
[23]  :end
[24]  :if 2=≡B ⋄ B←,⊂B ⋄ :end            A ensure B depth is ≥ 3
[25]  E←2>≡¨B ⋄ (E/B)←⊂¨E/B               A ensure each element is of depth 2
[26]  B←,B ⋄ '#'⎕wi'⍙wres'(0 0⍴0)        A define default result
[27]  :for I :in ⍳⍴B                     A for each element in argument
[28]      :if 'super '≡6↑P←↑I⊃B          A if argument starts with 'super '
[29]          E←(1+F←+/P ⎕ss'super')⊃D    A retrieve the super class name
[30]          G←⊂(5×F)↓P~' '             A method name
[31]          ⍎'A ',E,' G'                 A∇{VALUE:⎕error'Unknown class: ',E}
[32]      :else                          A else
[33]          ⍎'A ',C,' I⊃B'               A∇{VALUE:⎕error'Unknown class: ',C}
[34]      :end
[35]  :end
[36]  R←'#'⎕wi'⍙wres'                    A return result saved in ⍙wres
    ∇
```

As you can see, it is very easy to implement the **super** keyword in order to **superclass** methods in replacement of the standard class method.

Overloading Standard Properties

Another task you may want to perform for your new objects consists in extending some standard APL+Win properties, like the **properties** and **methods** properties.

It would be nice if we could use a **properties** property to know which user-defined properties are available for a given user-defined object and a **methods** property to ask about available methods.

For example we could define a `Qwi4` function that would work as follows:

```
      'ff' Qwi4 'New' 'Form11'
ff

      'ff' Qwi4 'properties'
ontop

       'ff' Qwi4 'methods'
New Move SetDefaultColor
```

We could try to implement the **properties** and **methods** properties into the class function (here `Form5`) as we did for the `ontop` property and the `Move` method. However that would mean we would have to implement them in ***every*** class function we develop, for every new object we invent. This would mean an important loss of time and room in the workspace.

Since we want **properties** and **methods** to be available properties for ***any*** object we create, we should find a way to implement these properties within the `Qwi` function itself. At first sight, it does not seem too complex when we notice that every user-defined object property or method has a corresponding `:case` statement in a class function. So we just need to write a program that will extract all lowercase names following each `:case` in the class function and we will have the list of properties we are looking for.

But then we are facing another difficulty which is engendered by inheritance: all the properties that a class knows are not necessarily defined in the class function pertaining to this class. Some (or all) of them might very well be defined in other classes from which this class inherits.

In order to best solve this problem, I have taken a design decision: this decision has been that the `∆class` user-defined data property should contain not only the object class, but the whole list of ancestor classes to this class. Let's take a few minutes explaining how this helps.

Example:

```
      'ff' ⎕wi 'Δclass'
Form11 Form10 Form5 Form

      ]display 'ff' ⎕wi 'Δclass'
.→-------------------------.
|.→-----..→-----..→----..→---.|
||Form11||Form10||Form5||Form||
|'------''------''-----''----'|
'∈------------------------------'
```

This means that *ff* is a **Form11** kind of a form which inherits from class **Form10** which itself inherits from class **Form5** which itself inherits from the standard APL+Win **Form** class.

The *Δclass* user-defined data property is recorded at each level of the class hierarchy by the *QwiRegister* function which I delayed discussing until now. Here it is:

```
    ∇ QwiRegister A;B;C;D;⎕io
[1]   ⍝∇ QwiRegister A -- Sets the object A Δclass property to B
[2]   ⍝∇ A ↔ object name
[3]   ⍝∇ Example: QwiRegister 'ff'
[4]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[5]
[6]   ⎕io←1                          ⍝ environement
[7]   B←A ⎕wi'class'                 ⍝ APL+Win class name
[8]   C←A ⎕wi'Δclass'                ⍝ class descent so far
[9]   :if C≡0                        ⍝ if Δclass not yet defined
[10]      A ⎕wi'Δclass'(C←⊂B)        ⍝ register APL+Win class name
[11]  :end                           ⍝ end
[12]  D←⊂(¯1+D⍳'[')↑D←⎕si[2;]        ⍝ calling function name (=UDC name)
[13]  A ⎕wi'Δclass'(D,C)             ⍝ append UDC name
    ∇
```

This function is easy to understand: any user-defined class must derive from a standard APL+Win class. At the time we create a new instance of class (with the *New* method), the *Δclass* property is not yet defined (i.e. is scalar 0) and therefore we register the APL+Win class from which our user-defined class derives (**line 10**). Then we append the user-defined class name (**line 13**) at this level of registration by simply getting the calling class function name (**line 12**).

Recording the whole class descent in variable *Δclass* is a great help in our search for the complete list of properties of a given class (including inherited properties). Because now, all we have to do is to write a *QwiProperties* which is able to retrieve the properties of one level and then loop calling *QwiProperties* at all levels of the class descent.

Here is the *QwiProperties* function:

```
    ∇ R←QwiProperties A;C;D
[1]   ⍝∇ R←QwiProperties A -- Return all properties from class A
[2]   ⍝∇ Example: QwiProperties 'Form5'
[3]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[4]
[5]   C←(⎕vr(,A)~' ')~' '              ⍝ get class function ⎕vr & remove blanks
[6]   D←C ⍷⎕ss ']:case'''              ⍝ find where :case statements occur
[7]   R←7↑¨D ⊂⎕penclose C              ⍝ partition ⎕vr accordingly
[8]   R←(∧\¨R≠'''')⌿⎕repl¨R            ⍝ select property (or method) names
[9]   R←(∈(↑¨R)∈'abcdefghijklmnopqrstuvwxyzΔ')/R  ⍝ keep only lowercase names
[10]  R←R~('.'∈¨R)/R                   ⍝ remove event properties
[11]  R←⊃R~((⊂'private')≡¨7↑¨R)/R       ⍝ remove private properties
[12]  R←R[⎕av⍋R;]                      ⍝ sort list of properties
    ∇
```

The comments in *QwiProperties* function make it self-explanatory.

Similarly we can define a *QwiMethods* utility, which reads:

```
    ∇ R←QwiMethods A;C;D
[1]   ⍝∇ R←QwiMethods A -- Return all methods from class A
[2]   ⍝∇ Example:  QwiMethods 'Form5'
[3]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[4]
[5]   C←(⎕vr(,A)~' ')~' '              ⍝ get class function ⎕vr & remove blanks
[6]   D←C ⍷⎕ss ']:case'''              ⍝ find where :case statements occur
[7]   R←7↑¨D ⊂⎕penclose C              ⍝ partition ⎕vr accordingly
[8]   R←(∧\¨R≠'''')⌿⎕repl¨R            ⍝ select property (or method) names
[9]   R←⊃(∈(↑¨R)∈'ABCDEFGHIJKLMNOPQRSTUVWXYZ')/R  ⍝ keep uppercase names
[10]  R←R[⎕av⍋R;]                      ⍝ sort list of methods
    ∇
```

The only remaining task is to change *Qwi3* to *Qwi4* as follows:

```
    ∇ R←A QWI4 B;C;D;E;F;G;I;P;⎕elx
[1]   ⍝∇ R←A QWI4 B -- ⎕WI emulator
[2]   ⍝∇ ⎕wself←'object_name'  QWI4 'New' 'class_name'
[3]   ⍝∇ ⎕wself←'object_name'  QWI4 'New' 'class_name'('prop1' value1)...
[4]   ⍝∇ res←  ['object_name'] QWI4 'prop'
[5]   ⍝∇       ['object_name'] QWI4 'prop' value
[6]   ⍝∇       ['object_name'] QWI4 ('prop1' value1)...('propN' valueN)
[7]   ⍝∇ res←  ['object_name'] QWI4 'Get_Method' [arguments]
[8]   ⍝∇       ['object_name'] QWI4 'Set_Method' [arguments]
[9]   ⍝∇ Copyright (c) 1997 Eric Lescasse [8mar97]
[10]
[11]  ⎕elx←'⍎HANDLERFOR ⎕DM'
[12]  :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end     ⍝ ⎕WSELF if no left arg
[13]  :if 1=≡B ⋄ B←,⊂B ⋄ :end            ⍝ be sure B is nested
[14]  :if 'New'≡↑B                       ⍝ if method is 'New'
```

```
[15]      C←2⊃B                          ⍝ get specified class name
[16]      B←(c'New' C),2↓B               ⍝ rebuild argument
[17]  :elseif 0≠⍴A ⎕WI'self'             ⍝ else if object A exists
[18]      :if 0≠C←↑D←A ⎕WI'⍙class'               ⍝ get registered class name
[19]          ⎕error'Use ⎕WI instead of QWI4'    ⍝ if none signal error
[20]      :end
[21]  :else                                      ⍝ if not 'New' & not an object
[22]      ⎕error'Unknown object: ',A             ⍝ signal error
[23]  :end
[24]  :if 2=≡B ⋄ B←,⊂B ⋄ :end            ⍝ ensure B depth is ≥ 3
[25]  E←2>≡¨B ⋄ (E/B)←⊂¨E/B              ⍝ ensure each element is of depth 2
[26]  B←,B ⋄ '#'⎕WI'⍙wres'(0 0⍴0)        ⍝ define default result
[27]  :for I :in ⍳⍴B                     ⍝ for each element in argument
[28]      :select P←↑I⊃B                 ⍝ test property/method name
[29]      :case 'class'                          ⍝ class property
[30]         '#'⎕WI'⍙wres' D                     ⍝ list of classes
[31]      :case 'methods'                        ⍝ methods property
[32]          E←''                               ⍝ special method names
[33]          R←D,[1.5](QWIMethods¨¨1↓D),⊂E      ⍝ get methods at each level
[34]         '#'⎕WI'⍙wres' R                     ⍝ save result
[35]      :case 'properties'                     ⍝ properties property
[36]          E←'class' 'methods' 'properties'   ⍝ special property names
[37]          E←⊃E,'?property' '?method'         ⍝ (continued)
[38]          R←D,[1.5](QWIProperties¨¨1↓D),⊂E   ⍝ get properties at each level
[39]         '#'⎕WI'⍙wres' R                     ⍝ save result
[40]      :else                                  ⍝ else
[41]          :if '?'=↑P                         ⍝ if property starts with ?
[42]              R←⍙ QWIDoc 1↓P                 ⍝ get property documentation
[43]             '#'⎕WI'⍙wres' R                 ⍝ save result
[44]          :elseif 'super '≡6↑P               ⍝ if argument starts with 'super '
[45]              E←(1↓F←↑/P ⎕NS'super')⊃D       ⍝ retrieve the super class name
[46]              G←⊂(5×F)↓P~' '                 ⍝ method name
[47]              ⍎'⍙ ',E,' G'                   ⍝∇{VALUE:⎕error'Unknown class: ',E}
[48]          :else                              ⍝ else
[49]              ⍎'⍙ ',C,' I⊃B'                 ⍝∇{VALUE:⎕error'Unknown class: ',C}
[50]          :end
[51]      :end
[52]  :end    ⍝
[53]  R←'#'⎕WI'⍙wres'                    ⍝ return result saved in ⍙wres
    ∇
```

When we query the methods property with `QWI4`, **lines 31 to 34** get executed and the `QWIMethods` is called for each of the object registered classes:

```
      'ff' QWI4 'methods'
Form11  New
        SetDefaultColor

Form10  New
        SetDefaultColor

Form5   Move
```

```
        New

Form
      ]display 'ff' Qwi4 'methods'
.+-------------------------.
+.+-----..+--------------.|
||Form11|+New             ||
|'------'|SetDefaultColor||
|        '---------------'|
|.+-----..+--------------.|
||Form10|+New             ||
|'------'|SetDefaultColor||
|        '---------------'|
|.+-----. .+---.          |
||Form5| +Move|           |
|'------' |New |          |
|        '----'           |
|.+----.  .e.             |
||Form|   | |             |
|'-----'  '-'             |
'e-------------------------'
```

Similarly, when we query the **properties** property with `Qwi4`, **lines 35 to 39** get executed and the `QwiProperties` is called for each of the object registered classes:

```
      'ff' Qwi4 'properties'
Form11
Form10
Form5   ontop

Form    class
        methods
        properties
        ?property
        ?method

      ]display 'ff' Qwi4 'properties'
.+--------------------.
+.+-----..+---.       |
||Form11|ϕ    |       |
|'------''----'       |
|.+-----..+---.       |
||Form10|ϕ    |       |
|'------''----'       |
|.+-----. .+-----.    |
||Form5| +ontop|      |
|'------' '------'    |
|.+----.  .+----------.|
||Form|   +class      ||
|'-----'  |methods    ||
|         |properties||
|         |?property ||
|         |?method   ||
|         '----------'|
'e--------------------'
```

This result is a 2-column nested matrix where the 1st column contains the class descent of the object and the second column contains the list of properties defined at each level.

Therefore we see that the ontop property is not defined at the **Form11** class level, neither in its **Form10** parent class, but only in its **Form5** grandparent class.

Looking at the methods tree, we see that the `SetDefaultColor` method is implemented at the **Form11** class level and at the **Form10** parent level. This way we know for example that we could use the **super** keyword to invoke the **superclass** `SetDefaultColor` method.

This really is great and very very powerful.

Few Object Oriented development systems allow you to have a global vision of an object inheritance tree and all properties and/or methods defined at each level of the tree.

> This is the kind of example which fully demonstrates the great power of APL compared to other systems. The great ease of use of nested arrays combined with the ability to visualize them in the APL session as you develop an application gives APL developers a winning advantage over any other development system. This needs to be kept in mind.

If you look carefully at the `Qwi4` version of the `Qwi` function above, you see that I have also included 2 other enhancements.

One is the class property (see **lines 29 & 30**):

```
      'ff' Qwi4 'class'
Form11 Form10 Form5 Form
```

This is simply a cover function for:

```
      'ff' ⎕wi 'Δclass'
Form11 Form10 Form5 Form
```

The second enhancement is the ability to retrieve a property or method documentation by prefixing it with a question mark (**lines 41 to 43**) which leads us to the next paragraph.

Auto-documenting Properties and Methods

We can go one step further with the `Qwi4` function and create a `Qwi5` function.

Following the new feature of the APL+Win 2.0 OCX interface which lets you query an OCX property or method documentation by prefixing it with a question mark (?), we would like to be able to do:

```
      'ff' Qwi4 'New' 'Form12'
ff

      'ff' Qwi4 '?ontop'
ontop Determines if the form is on top of all other forms or not
      Syntax: boolean ←'object' Qwi 'ontop'
              'object' Qwi 'ontop' boolean

      'ff' Qwi4 '?Move'
Move moves the form to a new location
     Syntax: 'object' Qwi 'Move' vertmove horzmove
     vertmove: 'top' 'center' 'bottom' or integer (% of screen height)
     horzmove: 'left' 'center' 'right' or integer (% of screen width)
```

The easiest way I found for auto-documenting our user-defined object properties and methods is to add comments just after their `:case` statement in their class function.

For example, the `Form5` class function has been modified to `Form12` as follows:

```
    ∇ A Form12 B;C;D;E;F;G;H;I;J;K;L;M;N;R;S;class;⎕io
[1]   ⍝∇ A Form12 B -- User Defined Form12 Class
[2]   ⍝∇ A ↔ object name
[3]   ⍝∇ B ↔ 'property'
[4]   ⍝∇    or 'property' value
[5]   ⍝∇    or 'Method'
[6]   ⍝∇    or 'Method' argument1 ... argumentN
[7]   ⍝∇ Requires: (F) QwiRegister
[8]   ⍝∇ Copyright (c) 1997 Eric Lescasse [8mar97]
[9]
[10]  ⎕io←1                                    ⍝ environment
[11]  C←⊃B                                     ⍝ property name or method name
[12]  D←1↓B                                    ⍝ property value or method arguments
[13]
[14]  :select C
[15]
[16]  :case 'New'                              ⍝ constructor
[17]  ⍝ Create a new instance of a Form12 object
[18]  ⍝ 'New' classname
[19]      A ⎕wi'Delete'                        ⍝ delete existing form
[20]      R←A ⎕wi 'New' 'Form'                 ⍝ create form
[21]       '⍎'⎕wi 'Awres' R                    ⍝ result returned in Awres property
```

```
[22]        A ⎕wi 'caption' 'APL+Win Form'    ⍝ default caption
[23]        A ⎕wi 'font' 'MS Sans Serif' 8    ⍝ default font
[24]        A ⎕wi 'scale' 5                   ⍝ pixel coordinates
[25]        QwiRegister A                     ⍝ register class name
[26]
[27]    :case 'Move'
[28]    ⍝ Moves the form to a new location
[29]    ⍝ 'Move' vertmove horzmove
[30]    ⍝ vertmove: 'top' 'center' 'bottom' or integer (% of screen height)
[31]    ⍝ horzmove: 'left' 'center' 'right' or integer (% of screen width)
[32]        :if 2≠⍴D ⋄ :return ⋄ :end                    ⍝ exit if not 2 arguments
[33]        (E F)←D                                      ⍝ get method arguments
[34]        G←(1+82=⎕dr E)⊃E 0                           ⍝ G is 0 if 1st arg char
[35]        H←(1+82=⎕dr F)⊃F 0                           ⍝ H is 0 if 2nd arg char
[36]        (I J)←'#'⎕wi'size'                           ⍝ get screen dimensions
[37]        (K L M N)←A ⎕wi'where'                       ⍝ get form position & size
[38]        '#'⎕wi'Δwres'(K L×2⍴'#'⎕wi'units')           ⍝ return old form position
[39]        K←(.01×0 50 100 G×I-M)['top' 'center' 'bottom'⍳E]  ⍝ new vert form pos
[40]        L←(.01×0 50 100 H×J-N)['left' 'center' 'right'⍳F]  ⍝ new horz form pos
[41]        A ⎕wi'where' K L M N                         ⍝ move form
[42]
[43]    :case 'ontop'                                    ⍝ ontop property
[44]    ⍝ Determines if the form is on top of all other forms or not
[45]    ⍝ boolean ← 'ontop'
[46]    ⍝ 'ontop' boolean
[47]        :if 0∈⍴D                                     ⍝ if not property value
[48]            '#' ⎕wi 'Δwres' (A ⎕wi 'Δontop')         ⍝ query property
[49]        :else                                        ⍝ else
[50]            S←(1+1=⍴D)⊃'hwnd_notopmost' 'hwnd_topmost'  ⍝ windows constant to use
[51]            S←S 0 0 0 0 'swp_nomove swp_nosize'       ⍝ other arguments
[52]            S←⎕wcall'SetWindowPos' (A ⎕wi'hwnd'),S   ⍝ call SetWindowPos
[53]            A ⎕wi 'Δontop' (1=⍴D)                    ⍝ register property value
[54]        :end
[55]
[56]    :else                                ⍝ process all other properties/methods
[57]        :if (⊂C)∈A ⎕wi'properties'       ⍝ if a standard property ...
[58]        :orif (⊂C)∈A ⎕wi'methods'        ⍝ or a standard method
[59]            :if 0∈⍴D                     ⍝ when no value or arguments passed
[60]                '#'⎕wi'Δwres'(A ⎕wi C)   ⍝ return current value
[61]            :else                        ⍝ else...
[62]                A ⎕wi (⊂C),D             ⍝ apply value or arguments
[63]            :end
[64]        :end
[65]
[66]    :end
    ∇
```

The utility which knows how to extract the comments relative to a given property or method is *QwiDoc* reproduced below:

```
    ∇ R←A QwiDoc E;F;G;H;I;J;K;L;M;N;P;Q;S;T;U;V;W;X;Y;Z;⎕io
[1]   ⍝∇ R←A QwiDoc E -- Return doc for object A property (or method) E
[2]   ⍝∇ Example: 'ff' QwiDoc 'ontop'
[3]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[4]
[5]   ⎕io←1                                          ⍝ environment
[6]   A←(,A)~' '                                     ⍝ object name
[7]   M←,⊂E←(,E)~' '                                 ⍝ property (or method) name
[8]   G←A ⎕wi'⍙class'                                ⍝ object class descent
[9]   :if 0∈⍴E                                       ⍝ if right argument is empty
[10]      M←(⎕split QwiProperties ↑G)~⊂' '           ⍝ find all properties
[11]      M←M,(⎕split QwiMethods ↑G)~⊂' '            ⍝ find all methods
[12]  :end                                           ⍝ end
[13]  U←1+⌈/⍴¨M                                      ⍝ length of longest property name
[14]  R←''                                           ⍝ default result
[15]  :for E :in M                                   ⍝ for each property or method
[16]      :for I :in G                               ⍝ for each level of inheritance
[17]          Z←⎕vr I                                ⍝ get class function ⎕vr
[18]          S←Z≠' '                                ⍝ find non blanks
[19]          F←S/Z                                  ⍝ ⎕vr with no blanks
[20]          Y←F ⎕ss']:case'''',E,''''              ⍝ find where property starts
[21]          :if 1∈Y                                ⍝ if found in ⎕vr
[22]              H←Y⍳1                              ⍝ get position where it starts
[23]              F←(H↓F)~'0123456789'               ⍝ drop what's before & remove numbers
[24]              J←F ⎕ss ⎕tcnl,'[]⍝'                ⍝ find lines starting with comments
[25]              K←F=⎕tcnl                          ⍝ find start of lines
[26]              L←+/∧\K/J                          ⍝ # of comment lines for this property
[27]              :if L=0 ⋄ :continue ⋄ :end         ⍝ continue if no documentation
[28]              F←((S\Y)⍳1)↓Z                      ⍝ ⎕vr starting at :case 'prop'
[29]              F←(F⍳⎕tcnl)↓F                      ⍝ ⎕vr starting at first comment
[30]              F←(¯1+(+\F=⎕tcnl)⍳L)↑F             ⍝ keep only desired comments
[31]              F←(F≠⎕tcnl)⊂F                      ⍝ line partition
[32]              F←(F⍳¨'⍝')↓¨F                      ⍝ return vector of comments
[33]              F←(+/¨∧\¨F=' ')↓¨F                 ⍝ remove leading blanks
[34]              :if 2≤⍴F                           ⍝ if there is a syntax comment
[35]              :andif 1∈(2≥F)⎕ss''''',E,''''      ⍝ if property found in line
[36]                  T←2≥F                          ⍝ extract it
[37]                  V←'Syntax: '                   ⍝ prepare syntax statement
[38]                  W←('←'∈T)⊃T⍳'←'                ⍝ is there a result?
[39]                  V←V,W↑T                        ⍝ catenate result
[40]                  V←V,''''object'' Qwi '         ⍝ catenate Qwi
[41]                  V←V,(+/∧\X=' ')↓X←W↓T          ⍝ catenate property & value
[42]                  (2≥F)←V                        ⍝ catenate texts
[43]              :end                               ⍝ end
[44]              :if 3≤⍴F                           ⍝ if there is a syntax comment
[45]              :andif 1∈(3≥F)⎕ss''''',E,''''      ⍝ if property found in line
[46]                  T←3≥F                          ⍝ extract it
[47]                  V←'          '                 ⍝ prepare syntax statement
[48]                  W←('←'∈T)⊃T⍳'←'                ⍝ is there a result?
[49]                  V←V,W↑T                        ⍝ catenate result
[50]                  V←V,''''object'' Qwi '         ⍝ catenate Qwi
```

```
[51]                V←V,(+/∧\X=' ')↓X←W↓T   ⍝ catenate property & value
[52]                (3⊃F)←V                 ⍝ catenate texts
[53]            :end                        ⍝ end
[54]            F←(⊂Up' '),¨F               ⍝ indentation
[55]            (U↑1⊃F)←U↓E                 ⍝ install property name
[56]            R←R,⊂F                      ⍝ catenate doc to result
[57]            :leave                      ⍝ leave :for loop if property found
[58]        :end                            ⍝ end if
[59]    :end                                ⍝ end classes loop
[60] :end                                   ⍝ end property/method loop
[61] R←∊(R,⊂⍬),¨⊂⎕tcnl),¨⊂⎕tcnl              ⍝ make result character vector
[62]
    ∇
```

Note that once again, this function has to deal with inheritance: it extracts a property or method documentation at the first level in the inheritance tree where this property or method is documented. If no documentation is available, the function returns an empty matrix.

So, it is very easy to document properties and methods: just add one or more comments in the class function, just after each `:case` statement. The first comment should describe the role of the property or method. The second (and possibly third) comment should describe the syntax: simply use comments like:

```
    boolean ← 'ontop'
or: 'object' boolean
```

which will be automatically converted by *QwiDoc* to:

```
Syntax: boolean ←'object' Qwi 'ontop'
        'object' Qwi 'ontop' boolean
```

Additional comments should be used to describe method arguments, especially those which must take their values in a list, like:

```
Syntax: 'object' Qwi 'Move' vertmove horzmove
vertmove: 'top' 'center' 'bottom' or integer (% of screen height)
horzmove: 'left' 'center' 'right' or integer (% of screen width)
```

Examples:

```
      'ff' Qwi4 'New' 'Form12'
ff

      'ff' Qwi4 '?Move'
Move Moves the form to a new location
     Syntax: 'object' Qwi 'Move' vertmove horzmove
     vertmove: 'top' 'center' 'bottom' or integer (% of screen height)
```

```
      horzmove: 'left' 'center' 'right' or integer (% of screen width)

      'ff' Qwi4 '?ontop'
ontop Determines if the form is on top of all other forms or not
      Syntax: boolean ←'object' Qwi 'ontop'
              'object' Qwi 'ontop' boolean


      'ff' Qwi4 '?'
ontop Determines if the form is on top of all other forms or not
      Syntax: boolean ←'object' Qwi 'ontop'
              'object' Qwi 'ontop' boolean

Move  Moves the form to a new location
      Syntax: 'object' Qwi 'Move' vertmove horzmove
      vertmove: 'top' 'center' 'bottom' or integer (% of screen height)
      horzmove: 'left' 'center' 'right' or integer (% of screen width)

New   Create a new instance of a Form12 object
      Syntax: 'object' Qwi 'New' classname
```

Closer to the Final Qwi Utility

There is one more enhancement we can bring to `Qwi4` in order to get closer to our final `Qwi` utility. We will enhance `Qwi4` to `Qwi5`.

Wouldn't it be nice if `Qwi` was aware of a system object (#) which would only represent our user-defined classes, so that we could query the following properties:

```
      '#' Qwi5 'properties'
 children classes properties

      '#' Qwi5 'classes'
 DemoForm Form1 Form10 Form11 Form12 Form2 Form3 Form4 Form5
       Form6 Form7 Form8 Form9 QwiNewClass

      '#' Qwi5 'children'
 ff

      'gg' ⎕wi 'New' 'Form'
gg
      '#' Qwi5 'children'
 ff

      'hh' Qwi5 'New' 'Form5'
hh
```

```
      '#' Qwi5 'children'
  ff hh
```

So `Qwi5` would only report information about user-defined classes and ignore standard APL+Win classes.

The next listing shows the new version of `Qwi` which implements a virtual system object with 3 properties (**properties, children** and **classes**).

```
    ∇ R←A Qwi5 B;C;D;E;F;G;H;I;J;K;⎕elx;⎕io
[1]   ⍝∇ R←A Qwi5 B -- ⎕WI emulator
[2]   ⍝∇ ⎕wself←'object_name'  Qwi5 'New' 'class_name'
[3]   ⍝∇ ⎕wself←'object_name'  Qwi5 'New' 'class_name'('prop1' value1)...
[4]   ⍝∇ res←  ['object_name'] Qwi5 'prop'
[5]   ⍝∇       ['object_name'] Qwi5 'prop' value
[6]   ⍝∇       ['object_name'] Qwi5 ('prop1' value1)...('propN' valueN)
[7]   ⍝∇ res←  ['object_name'] Qwi5 'Get_Method' [arguments]
[8]   ⍝∇       ['object_name'] Qwi5 'Set_Method' [arguments]
[9]   ⍝∇ Copyright (c) 1997 Eric Lescasse [8mar97]
[10]
[11]  ⎕io←1                                  ⍝ environment
[12]  ⎕elx←'⍎HANDLERFOR ⎕DM'                 ⍝ set error handling
[13]  :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end         ⍝ ⎕WSELF if no left arg
[14]  :if A='#'                              ⍝ if object is virtual system object
[15]      :select B                          ⍝ depending on right argument
[16]      :case 'classes'                    ⍝ if 'classes'
[17]          R←QwiClasses                   ⍝ return list of UDCs
[18]      :case 'children'                   ⍝ if 'children'
[19]          R←QwiChildren                  ⍝ return list of UDC instances
[20]      :case 'properties'                 ⍝ if 'properties'
[21]          R←'children' 'classes' 'properties'  ⍝ hard coded property list
[22]      :else                              ⍝ if any other property or method
[23]          R←0 0⍴0                        ⍝ return empty matrix
[24]      :end                               ⍝ end
[25]      :return                            ⍝ exit from function
[26]  :end                                   ⍝ end
[27]  :if 1=≡B←,B ⋄ B←,⊂B ⋄ :end             ⍝ be sure B is nested
[28]  :if 'New'≡↑B                           ⍝ if method is 'New'
[29]      C←2⊃B                              ⍝ get specified class name
[30]      B←(⊂'New' C),2↓B                   ⍝ rebuild argument
[31]  :elseif 0≠⍴A ⎕wi'self'                 ⍝ else if object A exists
[32]      :if 0≡C←↑D←A ⎕wi'⍙class'                ⍝ get registered class name
[33]          ⎕error'Use ⎕WI instead of Qwi5'     ⍝ if none signal error
[34]      :end                                    ⍝ end
[35]  :else                                       ⍝ if not 'New' & not an object
[36]      ⎕error'Unknown object: ',A              ⍝ signal error
[37]  :end                                   ⍝ end
[38]  :if 1=≡↑B ⋄ B←,⊂B ⋄ :end               ⍝ if method/prop with arg/value
[39]  :if 2=≡B ⋄ B←,⊂B ⋄ :end                ⍝ ensure B depth is ≥ 3
[40]  E←2>≡¨B ⋄ (E/B)←⊂¨E/B                  ⍝ ensure each element is of depth 2
```

```
[41]  '#'⎕wi'Awres'(0 0⍴0)                    ⍝ define default result
[42]  :for I :in ↑⍴B←,B                       ⍝ for each element in argument
[43]      K←' '⍪J←' ',1↓H←I⊃B                          ⍝ get method/property name
[44]      :select J←1↓(K∨1↓K,0)/J                      ⍝ test property/method name
[45]      :case 'class'                                ⍝ class property
[46]         '#'⎕wi'Awres' D                           ⍝ list of classes
[47]      :case 'events'                               ⍝ properties property
[48]          E←''                                     ⍝ special property names
[49]          R←D,[1.5](QwiEvents¨1↓D),⊂E              ⍝ get properties at each level
[50]         '#'⎕wi'Awres' R                           ⍝ save result
[51]      :case 'methods'                              ⍝ methods property
[52]          E←''                                     ⍝ special method names
[53]          R←D,[1.5](QwiMethods¨1↓D),⊂E             ⍝ get methods at each level
[54]         '#'⎕wi'Awres' R                           ⍝ save result
[55]      :case 'properties'                           ⍝ properties property
[56]          E←'class' 'methods' 'properties'         ⍝ special property names
[57]          E←⊃E,'?property' '?method'               ⍝ (continued)
[58]          R←D,[1.5](QwiProperties¨1↓D),⊂E          ⍝ get properties at each level
[59]         '#'⎕wi'Awres' R                           ⍝ save result
[60]      :else                                        ⍝ else
[61]          :if '?'=↑J                               ⍝ if property starts with ?
[62]              R←A QwiDoc 1↓J                       ⍝ get property documentation
[63]             '#'⎕wi'Awres' R                       ⍝ save result
[64]          :elseif 'super '=6↑J                     ⍝ if argument starts with 'super '
[65]              E←(1+F←+/J ⎕ss'super')⊃D             ⍝ retrieve the super class name
[66]              G←⊂(5×F)↓J~' '                       ⍝ method name
[67]    ⍎'A ',E,' G'  ⍝∇{VALUE:⎕error'Unknown class: ',E}{⍵:⎕error(∧\⎕dm=⎕tcnl)/⎕dm}
[68]          :else                                    ⍝ else
[69]    ⍎'A ',C,' H'  ⍝∇{VALUE:⎕error'Unknown class: ',C}{⍵:⎕error(∧\⎕dm=⎕tcnl)/⎕dm}
[70]          :end                                     ⍝ end
[71]      :end                                         ⍝ end
[72]  :end                                    ⍝ end
[73]  R←'#'⎕wi'Awres'                         ⍝ return result saved in Awres
      ∇
```

The function has also been cleaned up and improved over `Qwi4` in a few places.

The new lines dealing with the virtual system object (**lines 14 to 26**) are very simple to understand: they make use of the following subroutines:

```
      ∇ R←QwiChildren
[1]   ⍝∇ R←QwiChildren -- Returns the list of user defined class objects
[2]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[3]
[4]   R←'#'⎕wi'children'      ⍝ find all system object children
[5]   R←R~(0=¨R ⎕wi¨⊂'Aclass')/R ⍝ keep those which have a Aclass property
      ∇
```

and:

```
    ∇ R←QwiClasses;A;B;C;D
[1]   ⍝∇ R←QwiClasses -- Returns the list of available user defined classes
[2]   ⍝∇ Copyright (c) 1997 Eric Lescasse [10mar97]
[3]
[4]   B←(⎕crl⍠(C←⎕split ⎕nl 3),⍠'c'[0]')~⍠' ' ⍝ get line 0 of all ws fns
[5]   D←1∊⍠B ⎕ss⍠c';class'                    ⍝ find those having ;class
[6]   B←(⎕vr⍠A←(D/C)~⍠' ')~⍠' '               ⍝ get ⎕vr of those functions
[7]   C←1∊⍠B ⎕ss⍠c':case''New'''              ⍝ find those having :case 'New'
[8]   C←∊C∧1∊⍠B ⎕ss⍠c'QwiRegister'            ⍝ and those using QwiRegister
[9]   R←C/A                                  ⍝ get their names
    ∇
```

The comments in these functions should be enough to explain them.

The Final Qwi Utility

We can still enhance *Qwi5* so that we get our final *Qwi* utility.

Our goal is that *Qwi* be a self-contained utility and our goal is also to develop new classes of objects which would also be represented by self-contained APL+Win functions.

So we will make the following improvements within *Qwi5*:

- first, we will integrate all our *Qwi* subroutines (*QwiChildren*, *QwiClass*, *QwiClasses*, *QwiDoc*, *QwiMethods* and *QwiProperties*)
- then, we will add a new method to *Qwi*, called **Register**, which will correspond to the *QwiRegister* subroutine

This way, all you will need in a new clear workspace to start defining and exploiting your own new classes of objects will be **2 functions:**

```
Qwi
HANDLERFOR
```

Note that, when using *Qwi*, the instruction required to register the class of an object instance should be changed from:

```
QwiRegister A     ⍝ register class name
```

to:

```
A Qwi 'Register'  ⍝ register class name
```

Classes **Form1** to **Form12** will not run with *Qwi* because they use the former *QwiRegister* method (supported by *Qwi1* to *Qwi5* only) to register object classes. I assume that from now on you will prefer to use the self-contained *Qwi* function because it is more convenient, so you should get into the habit of registering object classes using the *'object' Qwi 'Register'* instruction.

So here is the code for the final self-contained *Qwi* utility:

```
    ∇ R←A Qwi
B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;S;T;U;V;W;X;Y;Z;QwiEvents;QwiMethods;QwiProperties;⎕
elx;⎕io
[1]   ⍝∇ R←A Qwi B -- ⎕WI emulator
[2]   ⍝∇ ⎕wself←'object_name'  Qwi 'New' 'class_name'
[3]   ⍝∇ ⎕wself←'object_name'  Qwi 'New' 'class_name'('prop1' value1)...
[4]   ⍝∇ res←  ['object_name'] Qwi 'prop'
[5]   ⍝∇       ['object_name'] Qwi 'prop' value
[6]   ⍝∇       ['object_name'] Qwi ('prop1' value1)...('propN' valueN)
[7]   ⍝∇ res←  ['object_name'] Qwi 'Get_Method' [arguments]
[8]   ⍝∇       ['object_name'] Qwi 'Set_Method' [arguments]
[9]   ⍝∇ Requires: (F) HANDLERFOR (only if error occurs)
[10]  ⍝∇ Copyright (c) 1997 Eric Lescasse [16mar97]
[11]
[12]  ⎕io←1                                          ⍝ environment
[13]  ⎕elx←'⍎HANDLERFOR ⎕DM'                         ⍝ set error handling
[14]  :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end                 ⍝ ⎕WSELF if no left arg
[15]  :if A≡'#'                                      ⍝ if object is system object
[16]      :select B                                  ⍝ depending on right argument
[17]      :case 'classes'                            ⍝ if 'classes'
[18]          C←⎕split ⎕nl 3                         ⍝ all fns
[19]          E←(⎕crl⍤C,⍤c'[0]')~⍤' '               ⍝ get line 0 of all ws functions
[20]          D←1∊⍤E ⎕ss⍤c';class'                   ⍝ find those having ;class
[21]          E←(⎕vr⍤F←(D/C)~⍤' ')~⍤' '              ⍝ get ⎕vr of those functions
[22]          C←1∊⍤E ⎕ss⍤c';case''New'''             ⍝ find those having :case 'New'
[23]          C←∈C∧1∊⍤E ⎕ss⍤c'Qwi''Register'''       ⍝ and those using QwiRegister
[24]          R←C/F                                  ⍝ get their names
[25]      :case 'children'                           ⍝ if 'children'
[26]          R←'#'⎕wi'children'                     ⍝ find all system object children
[27]          R←R~(0≠⍤R ⎕wi⍤c'∆class')/R             ⍝ keep those with a ∆class prop
[28]      :case 'properties'                         ⍝ if 'properties'
[29]          R←'children' 'classes' 'properties'    ⍝  hard coded property list
[30]      :else                                      ⍝ if any other property or method
[31]          R←0 0⍴0                                ⍝ return empty matrix
[32]      :end                                       ⍝ end
[33]      :return                                    ⍝ exit from function
[34]  :end                                           ⍝ end
[35]  :if 1=≡B←,B ⋄ B←,⊂B ⋄ :end                     ⍝ be sure B is nested
```

```
[36]  :if 'New'≡↑B                          ⍝ if method is 'New'
[37]      C←2⊃B                             ⍝ get specified class name
[38]      B←(⊂'New' C),2↓B                  ⍝ rebuild argument
[39]  :elseif 'Register'≡↑B                 ⍝ if method is 'Register'
[40]      E←A ⎕wi'class'                    ⍝ APL+Win class name
[41]      F←A ⎕wi'Δclass'                   ⍝ class descent so far
[42]      :if F≡0                           ⍝ if Δclass not yet defined
[43]          A ⎕wi'Δclass'(F←⊂E)           ⍝ register APL+Win class name
[44]      :end                              ⍝ end
[45]      G←⊂(¯1↓G⍳'[')↑G←⎕si[2;]           ⍝ calling fn name (=udc name)
[46]      A ⎕wi'Δclass'(G,F)                ⍝ append udc name
[47]      :return                           ⍝ exit after registration
[48]  :elseif 0≠⍴A ⎕wi'self'                ⍝ else if object A exists
[49]      :if 0≡C←↑D←A ⎕wi'Δclass'          ⍝ get registered class name
[50]          ⎕error'Use ⎕WI instead of ⎕wi' ⍝ if none signal error
[51]      :end                              ⍝ end
[52]  :else                                 ⍝ if not 'New' & not an object
[53]      ⎕error'Unknown object: ',A        ⍝ signal error
[54]  :end                                  ⍝ end
[55]  :if 1=≡↑B ⋄ B←,⊂B ⋄ :end              ⍝ if method/prop with arg/value
[56]  :if 2=≡B ⋄ B←,⊂B ⋄ :end               ⍝ ensure B depth is ≥ 3
[57]  E←2>≡¨B ⋄ (E/B)←⊂¨E/B                 ⍝ ensure each element is depth 2
[58]  '#'⎕wi'Δwres'(0 0⍴0)                  ⍝ define default result
[59]  :for I :in ↑⍴B←,B                     ⍝ for each element in argument
[60]      K←' '≠J←' ',1⊃H←I⊃B               ⍝ get method/property name
[61]      :select J←1↓(K∨1↓K,0)/J           ⍝ test property/method name
[62]      :case 'class'                     ⍝ class property
[63]         '#'⎕wi'Δwres' D                ⍝ list of classes
[64]      :case 'events'                    ⍝ properties property
[65]          E←''                          ⍝ special property names
[66]          F←⊂'R←⎕wiEvents A;C;D'
[67]          F←F,⊂'C←(⎕vr(,A)~'' '')~'' '''
[68]          F←F,⊂'D←C ⎕ss '']:case''''''''
[69]          F←F,⊂'R←7↓¨D ⎕penclose C'
[70]          F←F,⊂'R←(∧\¨R≠''''''''')⎕repl¨R'
[71]          F←F,⊂'R←(ε(↑¨R)∈''abcdefghijklmnopqrstuvwxyzΔ'')/R'
[72]          F←F,⊂'R←⊃(''.''∈¨R)/R'
[73]          F←⎕def⊃F                      ⍝ fix ⎕wiEvents local fn
[74]          R←D.[1.5](⎕wiEvents¨¯1↓D),⊂E  ⍝ get properties at each level
[75]         '#'⎕wi'Δwres' R                ⍝ save result
[76]      :case 'methods'                   ⍝ methods property
[77]          E←''                          ⍝ special method names
[78]          F←⊂'R←⎕wiMethods A;C;D'
[79]          F←F,⊂'C←(⎕vr(,A)~'' '')~'' '''
[80]          F←F,⊂'D←C ⎕ss '']:case''''''''
[81]          F←F,⊂'R←7↓¨D ⎕penclose C'
[82]          F←F,⊂'R←(∧\¨R≠''''''''')⎕repl¨R'
[83]          F←F,⊂'R←⊃(ε(↑¨R)∈''ABCDEFGHIJKLMNOPQRSTUVWXYZ'')/R'
[84]          F←F,⊂'R←R[⎕av⍋R;]'
[85]          F←⎕def⊃F                      ⍝ fix ⎕wiMethods local fn
[86]          R←D.[1.5](⎕wiMethods¨¯1↓D),⊂E ⍝ get methods at each level
```

```
[87]          '#'⎕wi'Δwres' R                     ⍝ save result
[88]        :case 'properties'                    ⍝ properties property
[89]            E←'class' 'methods' 'properties'  ⍝ special property names
[90]            E←⊃E,'?property' '?method'        ⍝ (continued)
[91]            F←⊂'R←QwiProperties A;C;D'
[92]            F←F,⊂'C←(⎕vr(,A)~'' '')~'' '''
[93]            F←F,⊂'D←C ⎕ss '']:case'''''''
[94]            F←F,⊂'R←7↓¨D ⎕penclose C'
[95]            F←F,⊂'R←(∧\¨R≠'''''''''')⎕repl¨R'
[96]            F←F,⊂'R←(∊(↑¨R)∊''abcdefghijklmnopqrstuvwxyzΔ'')/R'
[97]            F←F,⊂'R←R~(''.''∊¨R)/R'
[98]            F←F,⊂'R←⊃R~((⊂''private'')≡¨7↑¨R)/R'
[99]            F←F,⊂'R←R[⎕av⍋R;]'
[100]           F←⎕def⊃F
[101]           R←D,[1.5](QwiProperties¨¯1↓D),⊂E    ⍝ get properties at each level
[102]         '#'⎕wi'Δwres' R                     ⍝ save result
[103]       :else                                 ⍝ else
[104]           :if '?'≡↑J                        ⍝ if property starts with ?
[105]               M←,⊂E←(,E←1↓J)~' '            ⍝ property (or method) name
[106]               G←A ⎕wi'Δclass'               ⍝ object class descent
[107]               :if 0∊⍴E                      ⍝ if right argument is empty
[108]                   M←⎕split↑(A Qwi'properties')[1;2]  ⍝ find all class properties
[109]                   M←M,⎕split↑(A Qwi'methods')[1;2]   ⍝ append all class methods
[110]                   M←M~¨' '                  ⍝ remove extraneous blanks
[111]               :end                          ⍝ end
[112]               U←1+⌈/∊⍴¨M                    ⍝ length of longest property name
[113]               R←''                          ⍝ default result
[114]               :for E :in M                  ⍝ for each property or method
[115]                   :for I :in G              ⍝ for each level of inheritance
[116]                       Z←⎕vr I               ⍝ get class function ⎕vr
[117]                       S←Z≠' '               ⍝ find non blanks
[118]                       F←S/Z                 ⍝ ⎕vr with no blanks
[119]                       Y←F ⎕ss']:case''',E,'''' ⍝ find where property starts
[120]                       :if 1∊Y               ⍝ if found in ⎕vr
[121]                           H←Y⍳1             ⍝ get position where it starts
[122]                           F←(H↓F)~'0123456789' ⍝ drop what's before
[123]                           J←F ⎕ss ⎕tcnl,'[]⍝' ⍝ lines starting with comments
[124]                           K←F=⎕tcnl         ⍝ start of lines
[125]                           L←+/∧\K/J         ⍝ # of comment lines for this prop
[126]                           :if L=0 ⋄ :continue ⋄ :end ⍝ continue if no doc
[127]                           F←((S\Y)⍳1)↓Z     ⍝ ⎕vr starting at :case 'prop'
[128]                           F←(F⍳⎕tcnl)↓F     ⍝ ⎕vr starting at first comment
[129]                           F←(¯1+(+\F=⎕tcnl)⍳L)↑F  ⍝ keep only desired comments
[130]                           F←(F≠⎕tcnl)⊂F     ⍝ line partition
[131]                           F←(F⍳¨'⍝')↓¨F     ⍝ return vector of comments
[132]                           F←(+/¨∧\¨F=' ')↓¨F  ⍝ remove leading blanks
[133]                           :if 2≤⍴F          ⍝ if there is a syntax comment
[134]                           :andif 1∊(2⊃F)⎕ss'''',E,'''' ⍝ if prop found in line
[135]                               T←2⊃F         ⍝ extract it
[136]                               V←'Syntax: '  ⍝ prepare syntax statement
[137]                               W←('←'∊T)×T⍳'←' ⍝ is there a result?
```

```
[138]                           V←V,W↑T          ⍝ catenate result
[139]                           V←V,'''object'' Qwi '  ⍝ catenate Qwi
[140]                           V←V,(+/∧\X=' ')↑X←W↓T  ⍝ catenate property & value
[141]                           (2⊃F)←V          ⍝ catenate texts
[142]                       :end                 ⍝ end
[143]                       :if 3≤⍴F             ⍝ if there is a syntax comment
[144]                       :andif 1∈(3⊃F)⍷ss''''',E,'''' ⍝ if prop found in line
[145]                           T←3⊃F            ⍝ extract it
[146]                           V←'          '   ⍝ prepare syntax statement
[147]                           W←('←'∈T)×T⍳'←'  ⍝ is there a result?
[148]                           V←V,W↑T          ⍝ catenate result
[149]                           V←V,'''object'' Qwi '  ⍝ catenate Qwi
[150]                           V←V,(+/∧\X=' ')↑X←W↓T  ⍝ catenate property & value
[151]                           (3⊃F)←V          ⍝ catenate texts
[152]                       :end                 ⍝ end
[153]                       F←(⊂⍤0p' '),¨F       ⍝ indentation
[154]                       (U↑1⊃F)←U↑E          ⍝ install property name
[155]                       R←R,⊂F               ⍝ catenate doc to result
[156]                       :leave               ⍝ leave :for loop if prop found
[157]                   :end                     ⍝ end if
[158]               :end                         ⍝ end classes loop
[159]           :end                             ⍝ end property/method loop
[160]           R←∊(R,⊂''''⍷tcnl),¨⍷tcnl         ⍝ make result character vector
[161]           R←(-+/∧\⌽R=⍷tcnl)↓R              ⍝ remove trailing ⍷tcnls
[162]          '#'⍷wi'⍙wres' R                   ⍝ save result
[163]       :elseif 'super '≡6↑J                 ⍝ if argument starts with 'super '
[164]           E←(1+F←+/J ⍷ss'super')⊃D          ⍝ retrieve the super class name
[165]           G←⊂(5×F)↓J←' '                   ⍝ method name
[166]  ⍙'A ',E,' G'  ⍙∇{VALUE:⍷error'Unknown class: ',E}{⍺:⍷error(∧\⍷dm≠⍷tcnl)/⍷dm}
[167]       :else                                ⍝ else
[168]  ⍙'A ',C,' H'  ⍙∇{VALUE:⍷error'Unknown class: ',C}{⍺:⍷error(∧\⍷dm≠⍷tcnl)/⍷dm}
[169]       :end                                 ⍝ end
[170]    :end                                    ⍝ end
[171] :end                                       ⍝ end
[172] R←'#'⍷wi'⍙wres'                            ⍝ return result saved in ⍙wres
      ∇
```

You may be interested in comparing `Qwi5` and `Qwi`, studying how the `Qwi` subroutines (`QwiChildren`, `QwiClass`, `QwiClasses`, `QwiDoc`, `QwiMethods` and `QwiProperties`) have been integrated into `Qwi`.

Some of them have simply been pasted in at the right place using **Ctrl+G** in the editor: this is the case with `QwiClasses` (see **lines 18 to 24** in `Qwi`).

In a few cases though, we were using some of these `Qwi` subroutines with the each (¨) operator: this was the case with `QwiProperties` and `QwiMethods`. Rather than trying to paste their code into `Qwi` and then make complex changes to implement each everywhere in the pasted code, it is much easier to first define these functions as local functions into `Qwi` and then to use them as before.

For example, **lines 91 to 100** in `Qwi` are used to define a local copy of `QwiProperties` (we must then not forget to localize `QwiProperties` into **line 0** of `Qwi`).

Had we shown function `Qwi` at the start of this chapter, you might have been frightened by the amount of code! I have chosen instead to start from a relatively simple version of the `Qwi` function (`Qwi1`) which already helped solve part of the problem and then progressively complement it with additional functionality finally to reach the ultimate stage of the development of a utility where you can try to make it self-contained.

This chapter is a good example of how a utility should be built and here is a summary of the rules:

- try to first isolate the main fundamental task the utility should perform
- write a first simple version of the utility which tackles this problem
- test the utility fully
- then one after the other, add additional behaviour to the utility
- at each step, fully test the utility
- when adding new functionality, do not hesitate to write simple subroutines: APL is made for that; moreover simple subroutines are easier to write and to test; there is also a chance these subroutines could serve as utilities later, outside the context of your current development; therefore save them in your `UCMDOBJ` file
- when everything you wanted your utility to cover is coded, then test everything again
- finally, when you are really sure the utility and all its subroutines are bug-free, you can start integrating subroutines into your utility to make it a self-contained utility

It's not that self-contained utilities are better than ones which are not self-contained (and in fact the contrary is sometimes true), but it is very much easier to use a self-contained utility in many different contexts because you just have to remember and copy ***one*** unique function into your workspace.

`Qwi` may become so important for your Object Oriented APL+Win developments that it's important that it is as easy as possible to use. That's why we made it self-contained. But I did keep a copy of the `QwiChildren`, `QwiClass`, `QwiClasses`, `QwiDoc`, `QwiMethods` and `QwiProperties` functions in my `UCMDOBJ` file because I am pretty sure I could need them one day.

Conclusion

Using **Object Oriented Programming**, one can extend the power of APL+Win by building the new kinds of objects one needs, and easily use them in a similar manner as ⎕*WI*, with a utility called *Qwi*. Once new objects are developed and fully tested they are easy to use and fully re-usable forever.

But what's more we have seen how to easily exploit the basic concepts of OOP with our new objects:

- inheritance,
- polymorphism,
- overriding methods, calling superclass methods
- method/property overloading
- encapsulation

Encapsulation?

Yes, we have used encapsulation all throughout this chapter. Remember: think of your objects as being the APL class functions (**Form1**, **Form2**, ... **Form12**, etc.) which implement them. Look carefully: with the help of the *:select ... :case ... :case :end* control structure, all class **properties** and **methods** have been encapsulated into just one class function.

Therefore, we can say that an object **IS** an APL function which totally encapsulates all of its functionality (except when inheritance is used, since the class function could call a parent class function).

Since this document was published in the Monthly APL+Win Training of May 1997, many users around the world have started using *Qwi* and creating their own objects. Some have sent me a copy of their objects.

This *Qwi* technology is also available for Dyalog where it adapts as well.

More information about the **Monthly APL+Win Training Program** can be obtained from the Lescasse Consulting's Web site at `www.lescasse.com`.

Eric Lescasse
Lescasse Consulting (SARL)
18 Rue de la Belle Feuille
92100 Boulogne, France. Tel: (33) 1.46.05.10.76; fax: (33) 1.46.04.60.23

Armstrong Numbers and APL

by Joseph De Kerf

This paper originally appeared in BACUS 18.4, December 1996
reprinted by permission of the author

In a previous paper we treated *niven numbers* [1]. In this paper we treat *armstrong numbers* [2].

A positive integer or natural number *I* is an *armstrong number* if the sum of the *J*th powers of its digits is equal to the original number, *J* being the number of its digits:

I = a*J* + b*J* + c*J* ...

For instance, the integer 153 is an armstrong number as $1^3+5^3+3^3 = 153$. Starting with the armstrong number 1, let *F(N)* be the *N*th armstrong number. Armstrong numbers *F(N)* for *N* = 1(1)28 are listed in Table 1.

As *F(N)* strongly increases with *N*, the frequency of the armstrong numbers strongly decreases when *N* increases. The nine digits 1 ... 9 are armstrong numbers. There exists no armstrong number of two digits. And for *J*≥3, there exist at most two armstrong numbers; if *I* is an armstrong number, then *I*+1 is an armstrong number, if and only if the last digit of *I* is 0. Or, if *I* is an armstrong number, then *I*–1 is an armstrong number, if and only if the last digit of *I* is 1. Examples:

370, 371 and 24678050, 24678051

So far, however, no algorithm has been found to calculate *F(N)* directly from *N*. This means that, to calculate *F(N)*, the sequence of positive integers or natural numbers 1, 2, 3,... has to be checked until *N* armstrong numbers have been found.

N	*F(N)*	*N*	*F(N)*	*N*	*F(N)*	*N*	*F(N)*
1	1	8	8	15	8208	22	4210818
2	2	9	9	16	9474	23	9800817
3	3	10	153	17	54748	24	9926315
4	4	11	370	18	92727	25	24678050
5	5	12	371	19	93084	26	24678051
6	6	13	407	20	548834	27	88593477
7	7	14	1634	21	1741725	28	146511208

Table 1: *F(N)* for *N* = 1(1)28

Two APL user-defined functions *ARMSTRONG1* and *ARMSTRONG2* for calculating the sequence of armstrong numbers *F*(1), *F*(2), ... *F(N)* are given below:

```
     ∇Z←ARMSTRONG1 N;I;J
[1]   →(N≤0)/0,Z←⍳I←0
[2]  LAB:J←⍴⍕I←I+1
[3]   →(I≠+/((J⍴10)⊤I)*J)/LAB
[4]   →(N>⍴Z←Z,I)/LAB
     ∇

     ∇Z←ARMSTRONG2 N;I;J
[1]   →(N≤0)/0,Z←⍳I←0
[2]  LAB:J←1+⌊10⍟I←I+1
[3]   →(I≠+/((J⍴10)⊤I)*J)/LAB
[4]   →(N>⍴Z←Z,I)/LAB
     ∇

     ARMSTRONG1 16
1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474

     ARMSTRONG2 16
1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474
```

In line `[1]` it is checked if *N* is negative or 0, a counter *I* is set to 0, and the explicit result *Z* is set to the empty vector `⍳0`. In line `[2]`, the counter *I* is increased by 1 and the number of digits *J* of *I* is evaluated. In line `[3]`, using the function encode, the counter *I* is split into its digits and it is checked if the sum of the *J*th powers of those digits is equal to *I*, in which case *I* is an armstrong number. Finally, in line `[4]`, if the counter *I* is an armstrong number, *Z* is catenated with this counter, and the loop is closed when the shape of the explicit result *Z* is equal to *N* (or ⌈*N*). If *N* is negative or 0, the empty vector `⍳0` is returned. If *N* is positive, the vector of armstrong numbers *F*(1), *F*(2), ... *F*(⌈*N*) is returned.

N	*ARMSTRONG1*	*ARMSTRONG2*	*ARMSTRONGA*	*ARMSTRONGB*
0	0.8 ms	0.8 ms	0.8 ms	0.8 ms
4	2.3 ms	2.3 ms	2.2 ms	2.2 ms
8	3.8 ms	3.8 ms	3.7 ms	3.7 ms
12	116.0 ms	115.0 ms	111.0 ms	109.0 ms
16	2.97 sec	2.94 sec	2.86 sec	2.80 sec
20	2.98 min	2.91 min	2.87 min	2.77 min
24	55.30 min	54.00 min	53.40 min	51.70 min
28		average: about 13.5 hours		

Table 2: CPU times for $N = 0(4)28$

The difference between *ARMSTRONG1* and *ARMSTRONG2* lies in the program used to count the digits of *I*. In *ARMSTRONG1* this is done by evaluating the shape of the format of *I*. In *ARMSTRONG2* it is done by adding 1 to the floor of the common logarithm of *I*.

In *ARMSTRONG1* and *ARMSTRONG2*, the count *J* of the digits of *I* and checking if *I* is an armstrong number is done in two separate lines, `[2]` and `[3]`, for readability. Performance in CPU times may be improved slightly by substituting a one-liner for lines `[2]` and `[3]`. This is done in the user-defined functions *ARMSTRONGA* and *ARMSTRONGB*:

```
     ∇Z←ARMSTRONGA N;I;J
[1]   →(N≤0)/0,Z←⍳I←0
[2]  LAB:→(I≠+/((J⍴10)⊤I)*J←⍴⍕I←I+1)/LAB
[3]   →(N>⍴Z←Z,I)/LAB
     ∇

     ∇Z←ARMSTRONGB N;I;J
[1]   →(N≤0)/0,Z←⍳I←0
[2]  LAB:→(I≠+/((J⍴10)⊤I)*J←1+⌊10⍟I←I+1)/LAB
[3]   →(N>⍴Z←Z,I)/LAB
     ∇

     ARMSTRONGA 16
1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474

     ARMSTRONGB 16
1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474
```

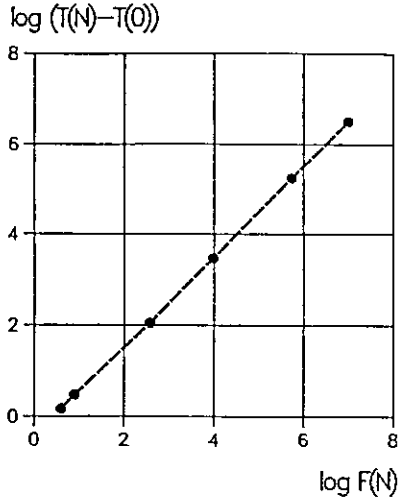


Figure 1: log(*T(N)*-*T*(0)) versus log *F(N)* for *N* = 4(4)24

To compare the effectiveness of the user-defined functions shown, CPU times *T(N)* for *N* = 0(4)28 have been monitored. Results are reported in Table 2 (*cf.* also Figures 1 and 2).

Of course *T(N)* - *T*(0) is approximately proportional to *F(N)*. Performance of *ARMSTRONGB* versus *ARMSTRONG*1 increases, for the domain investigated, from about 3% to 7% with *N*. Most importantly however, performance of the user-defined functions given is *very poor* and becomes impractical for even small values of *N*. Just as for the functions reported to calculate niven numbers, it would be a challenge to find an algorithm that *drastically* improves the performance of the functions to calculate the armstrong numbers.

The user-defined functions shown conform to the ISO Standard for APL [3]. Programming, calculations, and benchmarks have been done on a MicroLine Pentium-S 100, with Dyalog APL/W Version 7.1.2, under Windows 3.11 [4]. Benchmarks have been done using the system function ⎕*MONITOR*.

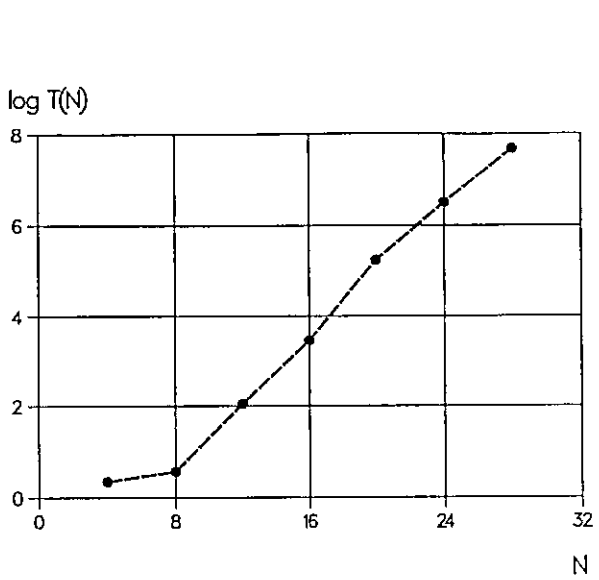


Figure 2: log *T(N)* versus *N* = 4(4)48

References

[1] J.De Kerf; *Niven Numbers and APL*; APL-CAM Journal, Vol. 18, No. 3, 16 September 1996, pp. 388-396

[2] D.D.Spencer; *Computers in Number Theory*; Computer Science Press, Rockville, Maryland, 1982, pp. 126-127

[3] International Standard ISO8485: 1989 (E), First Edition; 1989-11-01; *Programming Languages - APL*; International Organization for Standardization, Geneva, Switzerland, 1989.

[4] *Dyalog APL/W Language Reference - Version 7*; Dyadic Systems Ltd., Basingstoke, Hampshire, U.K., 1994.

Index to Advertisers

All queries regarding advertising in VECTOR should be made to Gill Smith, at 01439-788385, Email: apl385@compuserve.com.

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 (hardcopy with diskette as appropriate) to the Vector Working Group via:

Vector Administration, c/o Gill Smith
Brook House
Gilling East
YORK, YO6 4JJ
Tel: +44 (0) 1439-788385
Email: apl385@compuserve.com

Authors wishing to use Word for Windows should contact Vector Production for a copy of the APL2741 TrueType font, and a suitable Winword template. These may also be downloaded from the Vector web site at www.vector.org.uk

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.

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.

1997/98 Committee

Chairman	Anthony Camacho 0117-973 0036 100612.1057@compuserve.com	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Secretary	Ajay Askoolum 01737-771643 106173.3347@compuserve.com	42 Hanworth Road, Redhill, Surrey RH1 5HT
Treasurer	Nicholas Small 0181-980 7870 treas.apl@bcs.org.uk	8 Cardigan Road, LONDON E3 5HU
Journal Editor	c/o Vector Administration 01439-788385 apl385@compuserve.com	Gill Smith, Brook House, Gilling East YORK YO6 4JJ
Projects and Publicity	Dr Alan Mayer 01792-205678x4274 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
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
Administration	Rowena Small 0181-980 7870 treas.apl@bcs.org.uk	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Vacant Post	see above
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 (0121-3546550), Anthony & Sylvia Camacho, Ray Cannon (01252-874697), Marc Griffiths (01653-691745), Bob Hoekstra (01483 771028), Jon Sandles (01904-612882)

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

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