VECTOR

Advanced Programming Techniques for APL and J ...

The Journal of the British APL Association

A Specialist Group of the British Computer Society

ISSN 0955-1433
www.vector.org.uk
Vol.15 No.4 April 1999

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.

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

Membership Rates 1998-99

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

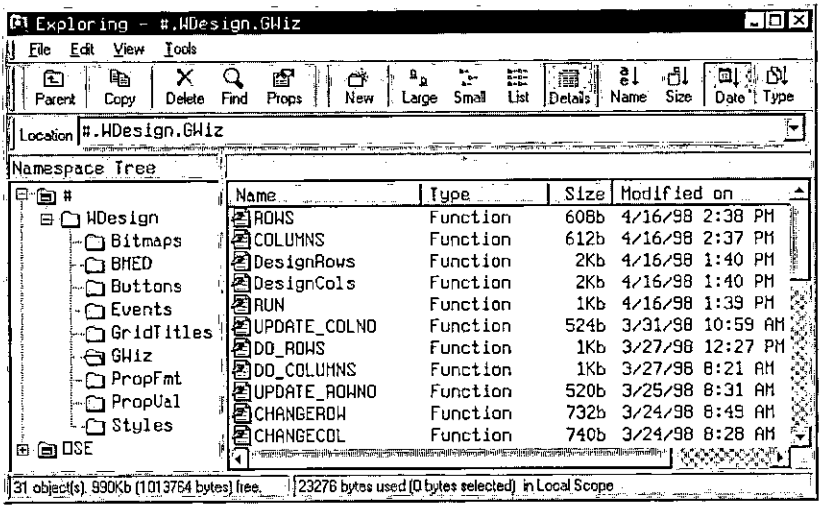
Advertisements in VECTOR should be submitted in typeset camera-ready format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates (excl VAT) are £250 per full page, £125 for half-page or less (there is a £75 surcharge per 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 YO62 4JJ. Tel: 01439-788385 Email: apl385@compuserve.com

Contents

dyalog APL

The Definitive APL for Windows™



Dyalog APL Namespaces let you ...

- Organise your workspace into **self-contained** sub-systems
- **Encapsulate** functions and variables *within* GUI objects
- **Isolate** utilities from your application code
- Construct **COM/DCOM** object hierarchies
- Support **multiple instances** in APL web server applications

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

MICROSOFT® WINDOWS™ COMPATIBLE

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

Microsoft is a registered trademark and Windows and the Windows Logo are trademarks of Microsoft Corporation

Editorial

"You have performed an illegal operation. You are under arrest"
Microsoft™... almost.

A software project like a football match, where the programmers are the players, why not? Personally I think there are a lot of reasons why the two disciplines don't have a lot in common, but on the other hand the parallel can be tried. Since we aim at big targets, let's think of a first division game. One first difference it's easy to find: no good programmer will ever even dream of earning anywhere near an average football player. And forget about Bill Gates, there can be only one like him and he won't leave his place very easily. Plus he has not been a programmer for many years now. Another difference is that programmers in general work like artists, composing another section of their great symphony every day, while soccer-players... well... just follow a ball and sometimes get to give it a kick. Is football at least a precise maybe even scientific game? No way. All they ever do is to throw the ball randomly hoping for some magic to happen while every minute they count their sweaty pounds accumulating in their bank accounts. What would be considered pure folly in many other ball sports (like basketball or volleyball) is simply accepted and sometimes even applauded as good play in football.

In the same way, computer programmers would never do anything that silly. Our objectives are always clear, the strategy to follow, the technology to be used, nothing is left to the hazard of a moment, everything is planned, the delivery precisely on time and the customer is always satisfied. We don't get a lot of fame, but fame is not always a good thing: imagine the shame of being spotted in a Seven Eleven shop while we buy our dinner, pre-cooked noodles and chips in a bag with beer in cans. Our reputation would be destroyed. Sometimes soccer players are hurt. Since technical directors and projects managers don't wear shoes with metal nails and don't spend hours in the gym, we have nothing to fear.

Now, what about a comparison between a movie♦ and a software project? It's immediately evident the larger freedom of a programmer not to mention the possibility of a version 2.0. A director only has one possibility. Have you ever heard of a movie called "Gone with the Wind version 2.2"? Has your favourite movie theatre ever shown "Fantasia Service Pack 3"? Would you be impressed

♦ For a different but somewhat analogous point of view on the same subject, please check the column "Developers in Love" in Vol.3 No. 4 (April 1999) of the "Microsoft Office and VBA Developer Magazine." Since the two articles were written in parallel, they did not influence each other. Let me quote: "... thinking about this superbly crafted, well-conceived film reminded me of how much I admire the work of anyone who excels in his or her chosen field...".

by a "Casablanca 97"? On the other hand, is the remake of "Sabrina" anywhere near as good as the original? Did any of you like "Highlander 2" or "Highlander 3" better than the original "Highlander"? In the world of cinema follow-ups are usually a disaster. In our wonderful cyberworld, versions onepointzero are hardly even worth the trouble, if not to get an idea of the future potential of the product. Our luck lies in the fact that not only are we given more than one chance to show what we can do, but in fact we are expected to use those extra chances. Moreover a director (and the actors) are not allowed any mistakes. Whenever a bug is found in one of our creations we can always promise "it will be fixed in the next release".

Finally, one may ask: what about the Oscar prize? That is definitely something a programmer does not normally get. Well, I am pleased to announce that a friend of mine has awarded, during a regular ceremony, the editorials of Vector with the NERV Oscar, for best production of the year. There you go.

Vector Back Numbers

Back numbers of Vector are available from:

British APL Association,
c/o Gill Smith,
Brook House, Gilling East,
YORK YO62 4JJ

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

Quick Reference Diary 1998–2000

Date	Venue	Event
May 21st	RSS, London	AGM and Vendor form - see p.6
August 10-14 1999	Scranton, Penn.	APL99
Late July 2000	Berlin	APL'00

The APL99 Conference is sponsored by SIGAPL and NYSIGAPL and is now in cooperation with ACM/SIGPLAN. The scope of the conference will be extended to include all array-processing languages including Mathematica, MATLAB, Array Based Fortran, SISAL and SAC. SIGPLAN members will be able to attend the conference at member rates. We are grateful to acknowledge the Arthur J. Kania School of Management at the University of Scranton for providing the conference facilities.

An electronic mailing list for information about APL99 has been set up. Subscription information is available on the web site

http://www.acm.org/sigapl/apl99.htm

We are tentatively planning to schedule pre-conference tutorials on Tuesday, August 10, 1999.

Dates for Future Issues of VECTOR

	Vol.16 No.1	Vol.16 No.2	Vol.16 No.3
Copy date	4th June	10th September	4th December
Ad booking	11th June	17th September	11th December
Ad Copy	18th June	24th September	18th December
Distribution	July 99	October 99	January 00

AGM & Vendor Forum - Friday May 21st
The RSS, Errol Street, London

2.00 — 2.15 pm: The British APL Association AGM

Hear about the latest news and meet the new committee. If you have any complaints about the way the association is being run — come and let your views be known.

2.15 — 5.00 pm: Vendor Forum

This year we have the two main vendors of APL development systems in the UK with all their latest news and features:

- Eric Baelen from APL2000, developers of APL+Win
- Peter Donnelly from Dyadic Systems, developers of Dyalog APL.

Tea and coffee will be available in the interval.

Venue

THE ROYAL STATISTICAL SOCIETY
12 Errol Street
London
EC1Y 8LX

(nearest tube — Barbican, Moorgate)

Contact

Places may be limited. To guarantee a seat please contact Jon Sandles on 01904-651063 (leave a message) or e-mail jon_sandles@csi.com. Details are subject to change — up to date information will be posted on http://www.vector.org.uk and comp.lang.apl

Vector's APL Nested Arrays Idiom Library

from Ajay Askoolum

In my editorial to 14.3 I hinted that Vector plans to organise issues by themes. The Journal Working Group has investigated the scope for this initiative and authorised a call for idioms on nested arrays and delegated the responsibility for co-ordinating this to me. I need your help.

All the prominent APL interpreters of today offer nested array primitives. At the outset, the implementations of the various vendors differed in approach but today the consensus is to comply with the IBM APL2 standard. The objective of this project is to compile an idiom library for nested array manipulation which will be common, mostly, to this standard.

I am asking for your particular contribution to this idiom library, in a format which includes a note on the purpose of your idiom, a workspace with a function encapsulating the idiom together with an example of its use. Ideally, the note could be in WORD, or if you do not have it, in WORDPAD or NOTEPAD which you will have as these are available on all versions of Windows. Any version of APL can be used.

Why would you want to contribute to this project? Well, I can offer three reasons:

1. Your contribution could be very valuable, share it with the APL community. You will be credited appropriately.

2. There is no APL material in print which deals with this aspect of APL: the treatment of the subject in the bible of APL is far from adequate for todays APL.

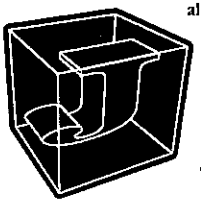
3. An idiom library of this nature may well engender a common approach to using nested arrays among the APL community. This is to your advantage in that it creates a common reference standard. If the idiom library is effective and is adopted, you will recognise coding on sight.

The NESTED APL IDIOM LIBRARY is only possible with your contribution.
Please make it happen.

Contributions may be sent to me via email or by post — contact details are on the inside back cover.

IT'S SURPRISING
HOW JSOFTWARE
MAKES YOUR DATA
JUMP OUT!

J for Windows 95/NT from Iverson Software Inc. provides everything you need to solve problems quickly. You can build complete stand alone Windows applications, or easily integrate J with other programs. For example, use J as an ActiveX control for a web page, an OLE server for Excel, or a 32-bit DLL called from a C program. J is also available on a variety of other platforms including Windows CE, Mac, Linux, Solaris and AIX. The language is identical in all versions, and programs not making use of platform-specific features will work unchanged on all systems.



J in a Box

Strand Software 19235 Covington Court Shorewood, MN 55331
Tel: (612) 470-7345 Fax: (612) 470-9202
www.jsoftware.com

News from Sustaining Members

APL2000

LEX2000 Inc. and its subsidiary APL2000 Announces Company To Be Acquired By Cognos

LEX2000 Inc. (OTC BB: LXTO), a leading provider of financial management software, and the parent company of APL2000 Inc, today announced that it has agreed to be acquired by COGNOS Incorporated (Nasdaq: COGNF). A definitive merger agreement has been signed and approved by the LEX2000 Inc. Board of Directors. The Agreement calls for payment of approximately US$10 million to be paid in a combination of cash and Cognos shares and is subject to LEX2000 satisfying certain terms and conditions prior to closing. The offer has been accepted by principal shareholders of LEX2000 who hold in excess of 75% of LEX2000 Inc. shares. The offer will be submitted to shareholders for approval.

LEX2000 Inc. designs, develops and markets financial tools and applications. It is best known for its budgeting, consolidation and management reporting software. Its APL2000 Inc subsidiary provides APL based tools upon which those applications are built..

In addition, SS&C today announced that its letter of intent relating to the acquisition of the APL2000 tools business of Princeton, NJ-based APL2000 Inc., a wholly owned subsidiary of LEX2000 Inc., of Atlanta, Georgia, has been terminated. Ottawa, Ontario-based Cognos(R) recently announced its acquisition of LEX2000. "Cognos is a premier company with the depth and expertise to advance the APL2000 language," stated Mr. Stone. President of SS&C. "We look forward to working with Cognos, and with the APL2000 team to continue to add strategic value to our PTS 2000(TM) client base."

LEX2000 is headquartered in Atlanta with offices in Princeton NJ, Westport CT, Bethesda MD. Additional information on the company can be found online at http://www.LEX2000.com or by calling (888) 534-7771. Additional information on APL2000 may be found at http://www.APL2000.com or by sending email to Sales@apl2000.com.

Dyadic Systems Ltd

Following the successful launch of Dyalog APL Version 8.2, Dyadic is pleased to announce the Dyalog APL Internet Software Publishing Initiative (APL/ISPI). This is a new Dyadic service for publishing applications written in Dyalog APL on the Internet.

APL/ISPI is based upon ActiveX technology, provided by Dyalog APL 8.2, that allows Dyalog APL applications to be packaged as ActiveX controls and downloaded and run in a web browser. APL/ISPI applications require a copy of DYALOG.DLL installed on the end-user's computer.

APL/ISPI applications will be hosted by a secure E-Commerce web server provided and run by Dyadic from their offices in Basingstoke, UK. End-users will purchase and download applications directly from this site or via links to it from other sites. End-users who do not already have DYALOG.DLL installed will receive it automatically as part of the download process. As a fully authorised Internet Software Publisher, Dyadic is able to deliver applications that are digitally signed and safe for scripting.

This service, which is effectively free for authors, offers Dyalog APL application developers a route to market and a distribution service for their software products. It will permit individuals and small companies to sell APL applications to a global customer base without the need for expensive advertising, production and distribution.

By openly publishing high-quality applications written in Dyalog APL, Dyadic intends to encourage the use of Dyalog APL for commercial applications and to attract new users to APL.

Dyadic believes that this initiative offers a tremendous commercial opportunity to APL developers. Please contact Dyadic for further details.

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, YO62 4JJ. Tel: 01439-788385, Email: apl385@compuserve.com

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	IBM RS/6000 MD320	11,736	APL POWERstation (Greyscale) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Greyscale Graph Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	13,817	APL POWERstation (Colour) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	22,656	Advanced APL POWERstation 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD520	37,114	APL POWERsystem (8-users) 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (2-8 user licence)
	IBM RS/6000 MD530	72,054	APL POWERsystem (16-users) 34.5 MIPS, 10.9 Mflops RISC Processor 32Mb RAM, 1.34Gb Disk, 2.3Gb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (8+ user licence)
	IBM RS/6000 MD540	122,842	APL POWERsystem (32-users) 41 MIPS, 13 Mflops RISC Processor 64Mb RAM, 1.7Gb Disk, 2.3Gb Tape CD-ROM Drive, 32 Ports AIX, Dyalog APL (8+ user licence)
Optima	IBM Compatible	poa	Complete networked or stand-alone solutions including configuration installation, maintenance and commissioning.

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
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	250	Upgrade to version 11 gives free runtime (£120 from any version)
	APL+Win v3.0	1350	A 32-bit Windows-hosted interpreter that runs under all Windows platforms including Windows 95. Note: Upgrade for £350 from version 1.8 or 2, £920 from version 1.0
	Migration to APL+Win	1060	from APL*PLUS/PC APL*PLUS/DOS any version.
			from earlier versions of APL*PLUS II
	APL+DOS	1250	APL*PLUS II DOS is renamed to APL+DOS.
	Migration to APL+DOS	760	from APL*PLUS/PC
	APL Link	200	Database Access
	APL Link Pro	500	
	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
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 Windows version 1.0	$1500	Product No. 5639-d46, part number 4229558.
	APL2 Runtime environment	$250	Part No. 39L8419 - Packager and runtime CDs. One additional runtime install $200, 5 additionals $900, 10 $1,500.
	APL2 for Sun Solaris	$1500	Product No. 5648-065.
	APL2 for AIX 6000	poa	Product No. 5765-012.
	APL2 Version 2	poa	Product No. 5688-228. Full APL2 system for S/370 and S/390
	APL2 Application Envt Vn2	poa	Product No. 5688-229. Runtime environment for APL2 packages
Insight Systems	APL*PLUS/PC	poa	APL systems marketed and supported ...
	Dyalog APL	poa	from: Dyadic, Manuglstics, IBM
	APL2	poa	under: Windows, OS2 and Unix
Iverson Software Inc.	J on the Web online registration ...		
	J Educational Edition	$95	
	J Standard Edition	$295	
	J Professional	$395	
	Books and accessories (discounts for reg users)		
	J Dictionary	$50	
	J User Manual	$50	
	J Phrases	$50	
	J Primer	$50	
	Concrete Math	$40	
	Exploring Math	$50	
	J User Conference Proceedings	$35	
	Mugs, T-shirts, Mousepads	$10 each	
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	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	poa	Master Production Scheduling
	FBS	poa	Forecasting and Budgeting System
	DRP	poa	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.
APL Group (see Eventra)			
APL Software\Services			
	APL Utilities	poa	DOS-based APL software for .AWS, .SAW, .TRY, . IWS, APL Conferences, utilities, unlocking (.AWS), and more. Send request for an email catalog to dick.holt@juno.com.
Beautiful Systems	ASF_FILE	$399	Dyalog APL/W auxiliary processor for access to APL*PLUS/PC APL component files (*.ASF).
	NAT_FILE	$299	Dyalog APL/W auxiliary processor which emulates the APL*PLUS/PC quad-N native file subsystem for access to the DOS file system.
	DBF_FILE	$299	Dyalog APL/W auxiliary processor for efficient block mode access to dBASE format files. Designed to get large amounts of data in and out of dBASE. Not suited for random access to small amounts of data (it does not handle keys).
	SF_READ	poa	Dyalog APL/W functions to read APL*PLUS data objects of any type or structure from *.SF style component files created by APL*PLUS II or III.
The Bloomsbury Software Company			
(for VSAPL)	Enhancements & Sharefile	poa	Component files, quad-functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	Compiler	poa	The First APL compiler!
(for APL2)	Sharefile/AP	poa	STSC's shared access component file system for APL2. Comparable to all APL*PLUS file systems: multi-user storage of APL2 arrays with efficient disk usage.
Causeway	*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: MS-DOS, LAN, Windows 3.1/95/NT. Ideal for APL customisation (APL*PLUS II and Dyalog APL); more than 150 installed.
Eventra	Qualedi	$1500-4000	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
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
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 5764-092
		$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	poa	Software management system for Dyalog APL (7.2 or later) based on hierarchical databases; includes gui interface and stand-alone functions. Pricing will be per-user. Available fall 1999.
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 APL	$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	FRESCO Business Graphics	poa	Fast and easy business graphics DLL
	AP126/PC	poa	GDDM interface for Dyalog APL/W
	AP127/PC	poa	ODBC interface for Dyalog APL/W
	AP119/PC	poa	TCP/IP interface for Dyalog APL/W
	FACS	poa	EMMA-like interface to DB2 or ODBC databases
	ISAP	poa	MS Windows DLL for calling Dyalog APL/W from web pages via MS Internet Information Server. Visit www.lingo.com for a 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.
Entropy Software	Consultancy	poa	Company reporting, business graphics, Windows applications with Dyalog APL/W
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.
First Derivative Analytics Ltd.	Consultancy	poa	Analysis, design, prototyping, development & testing of APL (especially financial) applications: Sharp, Dyalog APL/W.
General Software	Consultancy	from 200	Over 20 years experience with every version of APL, large mainframe systems and small PC based programmes.
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.
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 and ergonomics consultancy services.
	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.
Lingo Allegro	Consultancy	poa	General APL consulting, internet website development, migration and downsizing, performance tuning, education and training.

Mackay Kinloch Ltd	Consultancy	from £40/hr	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
MicroAPL	Consultancy	poa	Technical & applications consultancy.
Milinta Inc	Consultancy	poa	Design, development, maintenance, conversion, documentation in all APLs, most APs and some specific Sharp products (LOGOS, ViewPoint, Retrieve). Experience in multi-user, multi-task systems, databases, Y2K, Windows programming.
Ellis Morgan	Consultancy	250-500	Business Forecasting & APL Systems.
Oasis	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.
I-APL Ltd	Books	poa	I-APL stocks books written to go with the I-APL interpreter and some APL Press books. For a list write to 11 Auburn Road, Bristol BS6 6LS, ring 0117 973 0036 or email 100612.1057@compuserve.com.

Oasis	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.
ACM SIgAPL	International	APL QuoteQuad	Conferences; APL white pages; web site	$30
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	Vector provided	Mini-congress, APL ShareWare Initiative	
FinnAPL	Helsinki, Finland	FinnAPL Newsletter	Seminars on APL	100FIM(private), 30(student), 1000 (Co)
Japan APL Assoc	Tokyo	APL Journal	Monthly meetings (4th Sat)	10,000yen to join
NY SIgAPL	New York, USA	Big Apple APL	Monthly meetings	$35/$25(ACM)
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.
ACM SIgAPL	David Siegel	ACM, 1515 Broadway, 17th Floor, New York, NY 10036, USA (Subs only)
ADAPTA Software GmbH	Michael Baas	Marienhoehe 88, 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
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 YO62 4JJ, UK. Tel: 01439-788385 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
APL Group (see Eventra)		
APL Software\Services	Dick Holt	3802 N Richmond St, Suite 271, Arlington, VA 22207 USA Tel: +1 (703) 528-7624; Fax: +1 (703) 528-7617; Email: dick.holt@juno.org
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
Ajay Askoolum	Ajay Askoolum	42 Hanworth Road, Redhill, Surrey RH1 5HT Tel: 01737-771643 Email: 106173.3347@compuserve.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	PO Box 39, Mansfield, Ontario L0N 1M0 Canada Tel: +1-705-434-1239 Email: rjp@interlog.com
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) 886-2636; Fax: +1 (215) 886-4888
Bloomsbury Software	Peter Day	Bloomsbury House, 74-77 Great Russell St., London WC1B 3DA, 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: 0117-973 0036. email: 100612.1057@compuserve.com
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS, UK. Tel: 01252-874697 Email: 100430.740@compuserve.com
Causeway Graphical Systems Ltd	Adrian Smith	The Maltings, Castlegate, MALTON, North Yorks YO17 7DP, UK. Tel: 01653-696760 Fax: 01653-697719 Email: causeway@compuserve.com
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB, UK. Tel: 0171-404 5401. Compuserve: 100343,3210
Cinerea AB	Rolf Kornemark	Box 61, S-193 00 Sigtuna, Sweden. Tel/Fax: +46 859 255 421 Email rolf@cinerea.se
CODEWORK	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	18728 Bloomfield Road, Olney, MD 20832 USA Tel: +1 (301) 260-1435 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	37 Newhouse Rd, Bovingdon, Herts, HP3 0EJ, UK. Tel: 01442-834015
Eventra	Stuart Sawabini	Merritt Crossing, 440 Wheelers Farms Road, Milford, CT 06460, USA. Tel: +1(203) 882-9988. Fax: +1 (203) 882-9946 Email: ssawabini@eventra.com; eshaw@eventra.com
Evestic AB	Olle Evero	Bertellusvagen 12A, S-146 38 Tullinge, Sweden Tel&Fax: +46 778 4410 Email: olle.evero@mailbox.swipnet.se
FinnAPL	Olli Paavola	Suomen APL-Yhdistys RY, FinnAPL RF, PL 1005, 00101 Helsinki 10, Finland Email: olli.paavola@pyr.fi
First Derivative	Ken Chakahwata	114 Lemsford Lane, Welwyn Garden City, Herts AL8 6YP, UK
Analytics Ltd.		Tel/Fax: 01707-339620. Email: KenChakahwata@compuserve.com
General Software Ltd	M.E. Martin	Little Wester House, Westerhill Road, LINTON, Kent ME17 4BS Tel: 01622 749328 Fax: 01622 749365 E-mail: martin@gsoft.freeserve.co.uk
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	Dominique, Salisbury Road, Woking, Surrey, GU22 7UR, UK. Tel: 01483-771028 Email: bob.hoekstra@khamsin.demon.co.uk
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	IAC Human Interfaces, Unit R21, Auckland New Business Centre, St. Helen Auckland, Bishop Auckland, County Durham, DL14 9TX, UK. Tel: 01388-605544. Email: 100021,3073@compuserve.com
I-APL Ltd	Anthony Camacho (for queries, order forms)	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. Email: 100612.1057@compuserve.com
	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:alm@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
JAD Software	David Crossley	175 East 96th St., Apt. 17G, New York, NY 10128 Country: USA Tel: +1 (212) 369-8713 Fax: +1 (212) 751-0124
Japan APL Assoc	Toshio Nishikawa	1-8-13 Masujima Buld.6F Higashi Gotanda Shinagawa-ku, Tokyo Japan 141-0022. Tel: +81 (03) 3280-0411 Fax: +81 (03) 3280-0418 Email: KYY00361@niftyserve.or.jp
J Austria	Joachim Hoffmann	Bramley, Oldstead, YORK YO61 4BL. Tel: 01347-868121. Email: JoHo@ping.at
Mackay Kinloch Ltd	Alastair Kinloch	519 Webster's Land, Edinburgh EH1 2RX, Scotland, UK. Tel: +44 (0)131 228 5235 Email: akinloch@globalnet.co.uk
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:+1 800 546 4621-1 Email: sjhalasz@interaccess.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
Milinta Inc.	Dan Baronet	4215 St-Andre, Montreal, CANADA H2J 2Z3. Tel: +1 (514) 529-1434
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants GU32 3PE, UK. Tel: 01730-263843 Email: Ellis@mrtlfrm.demon.co.uk
NY Sig	David Siegel	PO Box 2697; New York, NY10163-2697, USA. Email: NYSIGAPL@ACM.ORG
Oasis b.v.	Theo Zwart, Louis Rijkse	Lekstraat 4, 3433 ZB Nieuwegein, Holland. Tel: +31 30 60 66 336 Fax: +31 30 60 65 844 Email: info@oasis.nl or rijkse@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
RadSys Technologies AB	Randolph Schrab	Lovsangarv. 18, S-756 52 Uppsala, Sweden. Tel: +46 18 32 41 53 Fax: +46 708 1996 11 Email: randolph.schrab@radsys.se
Renaissance Data Systems	Ed Shaw	P.O. Box 313, Newtown, CT 06470, USA. Tel: +1 (203) 270-9729 Email: rendata@apibooks.cnchost.com or orders@apibooks.cnchost.com
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 Miller	600 Park Avenue, Rochester, NY 14607-2926, USA. Tel: +1 (716) 271-1110. 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 Email: ashepp@compuserve.com
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)31483 Email:askom@obninsk.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: si@iti.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, UK. 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

ACM SigAPL	www.acm.org/sigapl/
Adaytum Software	www.adaytum.com/
AFAPL	www.ensmp.fr/~scherer/langlet/ (Journal available on line)
APL2000	www.APL2000.com/
APL-385	www.demon.co.uk/apl385/
AUSCAN	www.interlog.com/~rjp/auscan/
Capital PC User Group	http://cpcug.org/
Causeway	www.causeway.co.uk/
CODEWORK	www.inrete.it/cdwk/eng/homee.html
COSY (Bob Armstrong)	www.cosy.com/
Dinosoft Oy	www.dinosoft.fi/
Dyadic Systems Ltd	www.dyadic.com/
Eventra	www.eventra.com/
FinnAPL	www.pyr.fi/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/
Japan APL Association	www.naska.co.jp/JAPLA/
Lescasse Consulting	www.lescasse.com/
Lingo Allegro USA Inc	www.lingo.com/
MicroAPL Ltd	www.microapl.co.uk/
Oasis b.v.	www.oasis.nl/
Renaissance Data	www.aplbooks.cnchost.com/
RE Time Tracker Oy	www.rett.fi/
The Rochester Group Inc.	www.rochgrp.com/
Shepp & Associates	www.digitravel.com/
SigAPL	www.acm.org/sigapl/
Strand Software Inc.	www.jsoftware.com/
Rex Swain	www.pcnet.com/~rhswain/
Toronto SIG	www.torontoapl.org/
Jim Weigang	www.chilton.com/~jimw/

FTP SITES

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

Zark Newsletter Extracts

introduced by Jon Sandles

This section continues our series of extracts from the Zark APL tutor news. Once again, we publish the solution to last issue's exercise and also the solution to the crossword.

Because we have been playing catch-up with Vector over the last few months, I received two submissions relating to the Vector 15.2 Limbering reprint; I only received these as Vector 15.3 was going to press. The first from Mike Day:

> Jon
> Do you REALLY want entries for the Zark mat<==>vec question?
> In APL-I or II? Or using Dynamic Functions?
> Isn't it a bit old hat?
> Mike Day
> 8 2 99

The answer to the question about whether we want entries is, of course, yes! The problem may be old hat, but there are a lot of new approaches available to the array programmer these days (including new variants of APL - J and K, and also, as Mike mentions, Dyadic's dynamic functions) Also, APL is far from a dying language and there are a lot of new APL programmers out there. It does little harm to run through the optimum solutions to these problems one more time.

William R. Jones also sent in the following:

> Dear Mr. Sandles,
> I write in response to your "Zark Column" in Vector Vol.15 No.2. Although you are clearly seeking APL solutions to the task posed, perhaps you would also be interested in a J solution.
>
> Thus, I offer:
>
> ```
> mat_to_vec=. (#~ ~:&' ')@(,@,.)
> ```
>
> and
>
> ```
> vec_to_mat=. 4 : '(x.&{.)&.|: > <;._1 y.'
> ```

These J functions meet the conditions prescribed in your column. Their effects are illustrated by applications to the character matrix CM and the character vector CV as follows.

```
   ]CM=. 'red','green','blue',:'purple'
red
green
blue
purple
      '*' mat_to_vec CM
*red*green*blue*purple
   ]CV=. '/red/green/blue/purple'
/red/green/blue/purple
   '' vec_to_mat CV
red
green
blue
purple
   3 vec_to_mat CV
red
gre
blu
pur
```

Thank you for your consideration, Sincerely,
Bill Jones

jonesw@lafcol.lafayette.edu

If we compare this with the solutions printed in Zark all those years ago (and in Vector 15.3) you will notice that MAT_TO_VEC, like many of the solutions sent to Zark, gets an extra ounce of speed by deleting all blanks. This causes it to fail both when there are completely blank rows and also when the delimiter is blank. VEC_TO_MAT on the other hand is just 10% of the size of the winning APL solution and I asked Joachim Hoffman to compare it to the winning APL solution in terms of speed:

```
vec_to_mat=. 4 : '(x.&{.)&.|: > <;._1 y.'  NB. Original
NB. as tacit hook with take rank 1 - !!!!!!!!!!!!!!!!!!!!!!!!!!!
vtm=: {."1 ];._1
```

1. Summary: for small testdata: J is 7-times (0.5385 % 0.0715) faster than APL

```
   testvec=: GenTestVec 500
   $testvec
449617
A) Original J Version
   20 timex 'hhh=. (i.0) vec_to_mat testvec'
```

```
0.552
B) Short J Version
   20 timex 'hhh=. (i.0) vtm testvec'
0.0715                                                    NB. !!!!
C) Dyalog APL
   $ testvec
452904
   20 timex vec{delta}mat
0.5385
```

2. Summary for big testdata: J <vtm> can be approx. 13-times (3.2765 % 0.236) faster than <vec{delta}mat>.

```
testvec=: GenTestVec 1000
   $testvec
1946510
A) Original J Version
   20 timex 'hhh=:(i.0) vec_to_mat testvec'
2.1475
B) Short J Version
   20 timex 'hhh=. (i.0) vtm testvec'
0.236
C)Dyalog APL
$testvec
 1916077
20 timex vec{delta}mat
   3.2765
```

So JoHo's improved version of the J code has serious speed improvements over the APL version. I look forward to hearing further comments on this!

I hope you are all enjoying these reprints, and please remember, don't be afraid to submit a new solution!

Utility Corner: Entering Ranges of Numbers

(The purpose of this column is to make you more productive by introducing you to utility functions. Think of utility functions as APL functions that have names instead of symbols. By expanding your function vocabulary, you'll be able to write APL code that's more concise, more efficient, and more readable.)

You can use the APL take (↑) function to grab the first few (`3↑V`) or the last few (`¯3↑V`) elements of a vector. How can you grab the middle few?

Well, you can use the take function twice (`¯3↑13↑V`) or some combination of take and drop (`10↓13↑V` or `3↑10↓V`) or indexing (`V[10+⍳3]`).

How can you grab *several* middle subsets? One approach is to apply any of the methods above repeatedly, catenating the results. For example:

```
(3↑10↓V),(5↑20↓V),(2↑30↓V)
```

Another approach is to catenate the indices of the subsets, and then use indexing once. For example:

```
V[(10+⍳3),(20+⍳5),(30+⍳2)]
```

This latter method of referring to several subsets of a vector has two advantages over the former method. The first advantage is that less temporary storage is consumed during execution, and so it's more efficient. For example, the expression (3↑10↓*V*) generates the intermediate result (10↓*V*), which can be quite large. The expression *V*[10+⍳3], on the other hand, generates only three-element vectors as intermediate results.

The second advantage of the indexing approach is that it can be used for *assigning* subsets as well as selecting them. The take/drop approach can only select. (Note: APL2 does allow selective assignment for selection functions other than indexing, in the form (3↑10↓*V*)←Z.)

With the advent of replicate (/), a simple algorithm has emerged for generating at once all the indices required for selecting or assigning several subsets of a vector. Typically, the algorithm is buried in a subfunction called *PLUSIOTA*:

```
      10 20 30 PLUSIOTA 3 5 2
11 12 13 21 22 23 24 25 31 32
      (10+⍳3),(20+⍳5),(30+⍳2)
11 12 13 21 22 23 24 25 31 32
```

Here's a listing of the *PLUSIOTA* function:

```
     ∇ R←A PLUSIOTA B
[1]    ⍝ Returns (A[1]+⍳B[1]),(A[2]+⍳B[2])....
[2]    R←B/A-¯1↓0,+\B
[3]    R←R+⍳⍴R
     ∇
```

Take a minute to study this algorithm.

In last issue's LIMBERING UP column, we asked you to write two functions, *TO* and *UNTO*.

The function *TO* permits user input in the following form:

```
OFFICE NUMBERS:  13, 17 TO 25, 28 TO 31, 34 36
```

The *TO* function allows commas or blanks between numbers, and allows minus (-) or negative (¯) signs for negative numbers. Wherever the word "*TO*" appears, the set of continuous integers between the specified limits is inserted. If the second limit is smaller that the first limit (*e.g.* '9 *TO* 5'), the continuous integers are in descending order (9 8 7 6 5). For example:

```
      TO '3, 6 TO 9, 2 TO -2'
3 6 7 8 9 2 1 0 ¯1 ¯2
```

Normally, *TO* returns a numeric vector. However, if the character vector argument is poorly formed, the result is an integer scalar that indicates the problem.

Here are the possible error code results:

1	"*TO*" is missing arguments:	*N←TO '3 TO'*
2	Non-numeric values:	*N←TO '5 TO ONE'*
3	Non-integral values:	*N←TO '6.2 TO 8'*
4	Result is too long	*N←TO '1 TO 9E9'*

The last error code was suggested by a reader who recognised the possibility of a *WS FULL* from erroneous input. But what is too long? Let's define an optional left argument for the *TO* function. If given, the left argument is a number that represents the maximum length of the result of *TO*. For example:

```
      500 TO '10 11, 15 TO 1900'
4
      500 TO '10 11, 15 TO 190'
10 11 15 16 17 18 19 20 21 22 23 24 25 26
```

if the left argument is omitted or is empty (for those of you whose implementations of APL do not support ambivalent functions), a limit of 1000 is assumed:

```
      TO '10 11, 15 TO 1900'
4
      '' TO '10 11, 15 TO 1900'
4
```

Here's an illustration of the use of the *TO* functions:

```
[4]   ASK:⍞←P←'OFFICE NUMBERS: '
[5]    ONOS←TO(⍴P)⍎⍞
[6]    →OK IF 1=⍴⍴ONOS
[7]    →(E1,E2,E3,E4)[ONOS]
[8]   E1:'YOU OMITTED A NUMBER BEFORE/AFTER TO.'
[9]    →ASK
[10]  E2:'NUMBERS OR "TO" ONLY PLEASE.'
[11]   →ASK
[12]  E3:'WHOLE NUMBERS ONLY WITH "TO".'
[13]   →ASK
[14]  E4:'NO MORE THAN 1000 NUMBERS AT ONCE'
[15]   →ASK
[16]  OK: etc.
```

The *TO* function is listed below. If your version of APL does not support ⎕*VI* and ⎕*FI*, use instead the utility function Δ*VI*. Presented in the 1989 Quarter 4 issue of Zark APL Tutor News.

As you read through the *TO* function, keep your eyes open for the *PLUSIOTA* algorithm, and for the application of shifting techniques, both essential for writing *TO* as a nonlooping function.

```
     ∇ NVEC←LIM TO CVEC;E;M;N;R;RPL;S;T;⎕IO
[1]   ⍝ Converts character vector right argument
[2]   ⍝ into numeric vector, similar to ⎕FI.
[3]   ⍝ However, commas are replaced by blanks,
[4]   ⍝ minus signs (-) by negative signs (¯), and
[5]   ⍝ the work TO by the integers between the
[6]   ⍝ integers to the left and right of TO.
[7]   ⍝ For example:
[8]   ⍝
[9]   ⍝            TO '2, 6 TO 9, 2 TO -2'
[10]  ⍝       3 6 7 8 9 1 0 ¯1 ¯2
[11]  ⍝
[12]  ⍝ Assumes CVEC is a vector. LIM is a
[13]  ⍝ positive integer indicating the maximum
[14]  ⍝ length of the result (to avoid WS FULL).
```

```
[15]  ⍝ If empty or omitted, LIM←1000 is assumed.
[16]  ⍝ If the argument is poorly formed, returns
[17]  ⍝ a numeric scalar indicating the problem:
[18]  ⍝
[19]  ⍝ 1:  TO missing arguments   TO '3 TO'
[20]  ⍝ 2:  non-numeric values     TO '3 TO FOUR'
[21]  ⍝ 3:  non-integral values    TO '6.2 TO 7.15'
[22]  ⍝ 4:  too long               TO ' 1 to 900000'
[23]  ⍝
[24]   ⎕IO←1 ⍝ Origin 1 is fine
[25]  ⍝ Replace commas by blanks and '-' by '¯'
[26]   CVEC[(CVEC=',')/⍳⍴CVEC]←' '
[27]   CVEC[(CVEC='-')/⍳R]←'¯'
[28]   T←'TO'⍷CVEC ⍝ Flag occurrences of TO
[29]  ⍝ or: T←(CVEC='T')∧1↓(CVEC='O'),0
[30]   CVEC[0 1∘.+M←T/⍳R]←' ' ⍝ Blank out TOs
[31]   NVEC←2 ⍝ Result is 2 if non-numerics
[32]   →(∧/⎕VI CVEC)↓0 ⍝ Exit if this error
[33]   NVEC←R←⎕FI CVEC ⍝ Convert to numeric
[34]   →(0=⍴M)/0 ⍝ Exit if no TOs
[35]   N←CVEC≠' ' ⍝ Flag non blanks
[36]   N←N∧¯1↓1,~N ⍝ Flag 1st char of nonblnk grps
[37]   NVEC←1 ⍝ Result is 1 if 'TO' is missing args
[38]   T←(N∨T)/T ⍝ Which grps are 'TO'?
[39]   →(∨/(T,1)∧1,T)/0 ⍝ Exit if this error
[40]   T←T/⍳⍴T ⍝ Indices of TO grps
[41]   T←T-⍳⍴T ⍝ Inds into R of no.s preceding TO
[42]   S←R[T] ⍝ Starts of ranges
[43]   E←R[T+1] ⍝ ... and ends
[44]   NVEC←3 ⍝ Result is 3 if limits not integers
[45]   →(∨/(S≠⌈S)∨E≠⌈E)/0 ⍝ Exit if this error
[46]  ⍝ Set LIM to 1000 if not given or empty:
[47]   ⍎(2≠⎕NC'LIM')/'LIM←1000'
[48]   ⍎(0∈⍴LIM)/'LIM←1000'
[49]   RPL←(⍴R)⍴1 ⍝ init repl vec to expand/sqz vec
[50]   RPL[T]←M←|N←E-S ⍝ Amt to exp/sqz each range
[51]   NVEC←4 ⍝ Result is 4 of too long
[52]   →(LIM<+/RPL)/0 ⍝ Exit if this error
[53]   NVEC←RPL/R ⍝ Expand/sqz num vec
[54]   T←T++\¯1↓0,M-1 ⍝ Adjust inds for new vec
[55]  ⍝ Inds into result of TO values (except 1st):
[56]   R←M/T-¯1↓0,+\M
[57]   R←R+⍳⍴R
[58]  ⍝ Insert TO values:
[59]   NVEC[R]←(+\M/×N)+M/S-+\¯1↓0,M××N
      ∇
```

Shifting techniques are used on lines 29, 36, 39, 54 56, and 59. The *PLUSIOTA* logic is contained on lines 56 and 57. In place of these lines, you could have used

R←*PLUSIOTA M*. That is, the value of *R* is simply (*T*[1]+⍳*M*[1],(*T*[2]+⍳*M*[2]),(*T*[2]+⍳*M*[2]),(*T*[3]+⍳*M*[3]), ...

The most complicated line in the *TO* function is probably line 59. View the expression on that line as a modification of the *PLUSIOTA* algorithm:

```
PLUSIOTA:  (⍳+/M)   + M/S-+\¯1+0,M
 Line 59: (+\M/×N) + M/S-+\¯1+0,M××N
```

Notice these two algorithms produce identical results when ×*N* is all 1s. The modification is required to handle descending runs, as well as ascending runs. The result of the modified algorithm is equivalent to:

```
(S[1]+(⍳M[1])××N[1]), (S[2]+(⍳M[2])××N[2]), ...
```

which could be called the *PLUSORMINUSIOTA* algorithm.

Here's a utility function, in case you need it:

```
      ∇ R←A PLUSORMINUSIOTA B;M;S
[1]    ⍝ Returns (A[1]+(⍳B[1])××B[1]),
[2]    ⍝         (A[2]+(⍳B[2])××B[2]), ...
[3]     M←|B
[4]     S←×B
[5]     R←(+\M/S)+M/A-(S×-⎕IO)++\¯1+0,B
      ∇
```

The (*S*×-⎕*IO*) term is included so the function produces correct results for either index origin.

```
      10 20 30 PLUSORMINUSIOTA 3 ¯5 2
11 12 13 19 18 17 16 15 31 32
      (10+⍳3),(20-⍳5),(30+⍳2)
11 12 13 19 18 17 16 15 31 32
```

The *UNTO* function does the opposite of the *TO* function. For example:

```
      UNTO 3 6 7 8 9 2 1 0 ¯1 ¯2
3, 6 TO 9, 2 TO -2
```

The right argument of *UNTO* is a numeric vector. The result is a character vector. It should always be true that:

```
      NV≡TO UNTO NV
```

For every set of continuously increasing or decreasing integers that span at least four numbers, *UNTO* replaces all but the extreme numbers of the span by the word *TO*. For example:

```
      UNTO 5 6 7 8 9
5 TO 9
      UNTO 5 6 7
5 6 7
```

UNTO also inserts commas before and after "*TO*" phrases (except at the beginning or ending of the result) and replace negative signs (¯) by minus signs (-). For example:

```
      UNTO 3 2 1 0 ¯1 ¯2 ¯4 ¯5 ¯10 ¯9 ¯8 ¯7 ¯6
3 TO -2, -4 -5, -10 TO -6
```

The *UNTO* function is listed below. Again, look for the *PLUSIOTA* algorithm, and for the application of shifting techniques.

```
     ∇ CVEC←UNTO NVEC;C;D;I;L;R;RPL;S;⎕IO
[1]   ⍝ Converts numeric vector right argument into
[2]   ⍝ character vector, similar to monadic ⍕.
[3]   ⍝ However, runs of integers (4 or more long)
[4]   ⍝ are replaced by TO and commas, and
[5]   ⍝ negative signs (¯) by minus signs (-).
[6]   ⍝ For example:
[7]   ⍝
[8]   ⍝              UNTO 3 6 7 8 9 2 1 0 ¯1 ¯2
[9]   ⍝       3, 6 TO 9, 2 TO ¯2
[10]  ⍝
[11]  ⍝ Assumes NVEC is a vector. Uses global ⎕PP
[12]  ⍝ when formatting via ⍕.
[13]   ⎕IO←1 ⍝ Origin 1 is fine
[14]   R←⍴D←(1↓NVEC)-¯1↓NVEC ⍝ First difference
[15]  ⍝ Flag end of each run (use ⍴ in case empty):
[16]   I←R⍴((1↓D)≠¯1↓D),1
[17]   I←I/⍳R ⍝ Convert to indices
[18]   L←I-¯1↓0,I ⍝ Lengths of runs (less 1)
[19]  ⍝ indices of integer runs 4 long (diff is 3)
[20]   S←((L≥3)∧(1=|D[I])∧NVEC[I]=⌈NVEC[I])/⍳⍴I
[21]   →(×⍴S)/L1 ⍝ Branch if some, else format
[22]   CVEC←⍕NVEC
[23]   CVEC[(CVEC='¯')/⍳⍴CVEC]←'-' ⍝ Rpl '¯' by '-'
[24]   →0 ⍝ Done
[25]  L1:L←L[S]-1 ⍝ Lengths if such runs (less 2)
[26]   I←I[S]-L ⍝ Indices of no.s starting runs
[27]  ⍝ Squeeze out values within runs:
[28]   RPL←(⍴NVEC)⍴1
```

```
[29]    R←L/I-¯1↓0,+\L
[30]    RPL[R+⍳ρR]←0 ⍝[(I[1]+⍳L[1]),(I[2]+⍳L[2]),...]
[31]    NVEC←RPL/NVEC ⍝ Remove middle no.s
[32]    I←I-¯1↓0,+\L ⍝ Adjust inds for sqzd values
[33]    R←ρCVEC←⍕NVEC ⍝ Global: ⎕PP
[34]    CVEC[(CVEC='¯')/⍳R]←'-' ⍝ Rpl '¯' by '-'
[35]  ⍝ Indices of no.s to be followed by commas
[36]  ⍝ (before and after TO phrases):
[37]    C←(,I∘.+¯1 1)~I,0,ρNVEC
[38]  ⍝ Bld repl vec to expand for commas and TOs:
[39]    RPL←Rρ1
[40]  ⍝ Indices of space after each fmttd no.:
[41]    S←(CVEC=' ')/⍳R
[42]    RPL[S[C]]←2 ⍝ For commas
[43]    RPL[S[I]]←4 ⍝ For TO
[44]    CVEC←RPL/CVEC ⍝ Expand
[45]  ⍝ Adjust indices for expansion:
[46]    S←S+(0,+\RPL-1)[S]
[47]    CVEC[S[C]]←',' ⍝ Insert commas
[48]    CVEC[S[I]∘.+1 2]←((ρI),2)ρ'TO' ⍝ ... and TOs
      ∇
```

Shifting techniques are used on lines 14, 16, 18, 29, and 32. The *PLUSIOTA* logic is contained on lines 29 and 30.

Notice the mention of ⎕*PP* (print precision) on lines 11 and 33. Since the *UNTO* function uses monadic ⍕ (format), it is sensitive to the global setting of that system variable. For example:

```
      ⎕PP
10
      UNTO (⍳5),÷⍳5
1 TO 5, 1 0.5 0.3333333333 0.25 0.2
      ⎕PP←6
      UNTO (⍳5),÷⍳5
1 TO 5, 1 0.5 0.333333 0.25 0.2
```

The *UNTO* function illustrates a broad range of APL primitive functions. It's well worth your while to read through it. Experiment in immediate execution mode with those expressions you don't understand. If you're having trouble with *UNTO*, do the following:

1. Type in the function
2. Put stops on every line via:

```
(⍳100) ⎕STOP 'UNTO'  or SΔUNTO←⍳100
```

3. Run the function with this argument:

```
UNTO 3 6 7 8 9 5 6 0 1 2 3 2 1 0 ¯1 ¯2
```

4. Resume execution line by line via: →⎕LC. Look at the variables after each line.

5. Remove the stops via:

```
0 ⎕STOP 'UNTO' or SΔUNTO←0
```

LIMBERING UP: Tree Diagrams

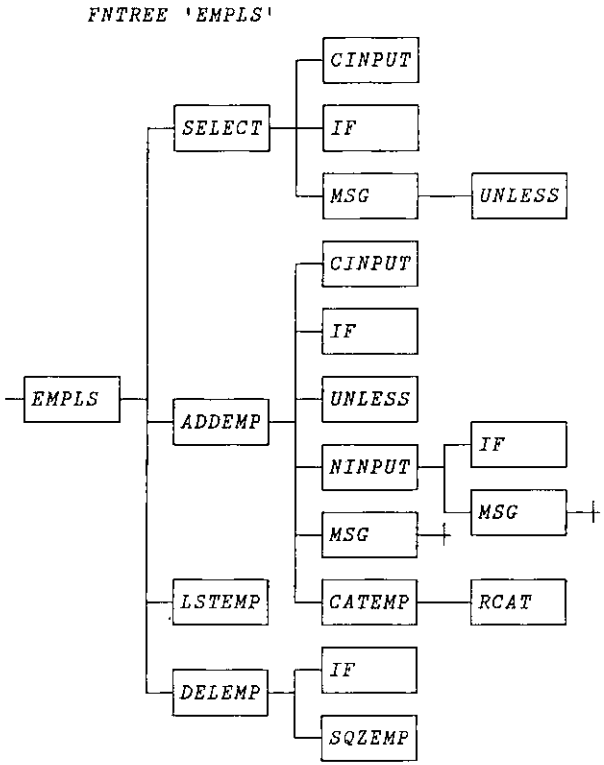
(The purpose of this column is to work some flab off your APL midsection. Like muscles, your APL skills can atrophy if not exercised with adequate frequency and variety. This column presents a task for you to perform. Set aside a few minutes from your busy schedule and work the task. Mail in your solution and stay tuned for the results.)

The mail we received in response to last issue's LIMBERING UP column was disappointing. Perhaps the topic was not sufficiently interesting or challenging.

This issue's topic should be both.

Suppose you've inherited an APL workspace from a fellow worker. As the new custodian of the workspace, you'll need to become familiar with its contents so you can make future fixes and enhancements.

You've been told that the name of the primary function in the workspace *EMPLS*. So, you load the workspace, copy the *FNTREE* function from your trusty workspace of utility functions and type



Wow! In a flash, you've got a pretty good idea of the flow of functions and subfunctions in the workspace. The *EMPLS* function calls *SELECT*, *ADDEMP*, *LSTEMP*, and *DELEMP*. The *SELECT* function, in turn, calls *CINPUT*, *IF*, and *MSG*. And so on. Now you can tack the function tree diagram on your bulletin board, sit back and start reading code.

(The ┼ symbol after the *MSG* function tells you that the subfunction details of the *MSG* function have been described already, above.)

Your task is to write the *FNTREE* function.

But not all at once. You can do it over two issues. In this issue, you need to design and write (and test!) a function named *TREE* that draws trees like the one above.

The *TREE* function assumes the existence of three global variables: *<names>*, *<relations>*, and *<shown>*. *<names>* is a character matrix of the names of

the functions in the tree. *<relations>* is a two column matrix of indices into *<names>*, that describes the relations between the functions; the first column is the index of the calling function, the second column is the index of the subfuction. *<shown>* is a bit vector with one element per row of *<names>*, telling which functions have already had their subfunctions shown. *<shown>* is both used and updated by *TREE*.

Oh, one more thing. *TREE* must be recursive. That is, it must call itself.

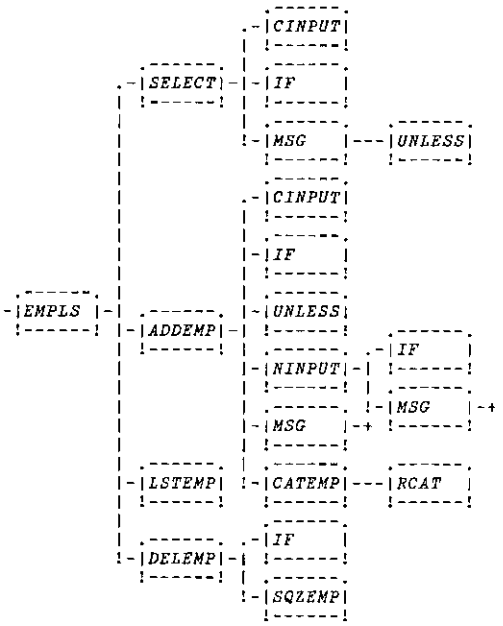
The above diagram is generated by the *TREE* function when given the following variables:

```
      names
 EMPLS
 SELECT
 CINPUT
 IF
 MSG
 UNLESS
 ADDEMP
 NINPUT
 CATEMP
 RCAT
 DELEMP
 SQZEMP
 LSTEMP_
      shown
0 0 0 0 0 0 0 0 0 0 0 0 0 0
      relations
 1  2
 2  3
 2  4
 2  5
 5  6
 1  7
 7  3
 8  4
 8  5
 7  4
 7  6
 7  8
 7  5
 7  9
 9 10
 1 13
11  4
11 12
 1 11
```

In the diagram above, we've used the special line-drawing characters available on the IBM PC. If you're using APL*PLUS PC or APL*PLUS II, you can place these characters on your function keys with the following function:

```
      ∇ PFKEYS
[1]     218 ⎕PFKEY 1 ⍵ ┌
[2]     191 ⎕PFKEY 2 ⍵ ┐
[3]     179 ⎕PFKEY 3 ⍵ │
[4]     196 ⎕PFKEY 4 ⍵ ─
[5]     195 ⎕PFKEY 5 ⍵ ├
[6]     180 ⎕PFKEY 6 ⍵ ┤
[7]     194 ⎕PFKEY 7 ⍵ ┬
[8]     193 ⎕PFKEY 8 ⍵ ┴
[9]     192 ⎕PFKEY 9 ⍵ └
[10]    217 ⎕PFKEY 10 ⍵ ┘
[11]    197 ⎕PFKEY 11 ⍵ ┼
      ∇
```

If your machine cannot draw these special characters, use the available APL symbols:



The *TREE* function should be as efficient as possible, as small as possible, and as readable as possible. Mail your solution to:

Jon Sandles
138 Burton Stone Lane
York
YO30 6DF
UK
e-mail: jon_sandles@csi.com

Reprinted with kind permission from Zark APL Tutor News, a quarterly publication of Zark Incorporated, 23 Ketchbrook Lane, Ellington, CT06029, USA

Challenge: "Design a utility to produce a function calling tree under Dyalog APL/W"

issued by Ray Cannon

One of the first utilities I wrote under Dyalog APL/W produced a calling tree for a specified function. Use of the Dyalog system function ⎕*REFS* makes the task easy, and this formed the basis of the code used in the "UCMD" utility "COPYUTILS" (Ref.: *Vector* 15.1 and http://www.vector.org.uk).

However, this code was written before namespaces were introduced, and now needs to be updated to take them into account.

The challenge then is to write a utility that, given the name of top function as its argument, returns a list of all the functions called from that point down (including the top function).

It should be able to handle simple explicit calls to functions in other namespaces (via the <name space path>.<fn name> syntax), but cannot be expected to handle run time variable calls (*e.g.* Function name resolution via the ⎕*PATH* system function, or via either the monadic or dyadic execute).

Each name in the calling tree should be qualified with its namespace path, either relative to the top function or as an absolute path. By this I mean that if the top function argument is AA and it calls BB and UTIL.CC, and BB calls #.ZZ the result would be:

AA
BB
UTIL.CC
#.ZZ

If the argument was given as #.app.AA, then the result would be

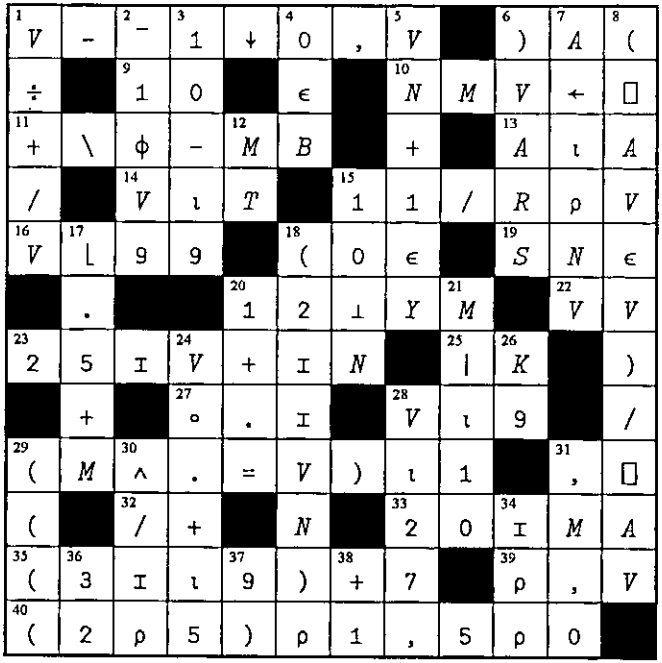
#.app.AA
#.app.BB
#.app.UTIL.CC
#.ZZ

It must be able to handle recursive calls and should only list a called object once.

The main problem I have had in trying to write this code is that the □*REFS* system function can no longer be used as it does not treat a function name (which includes namespace path information) as a single reference. Thus for example, a call "#.app.BB" produces two references "app" and "BB", and it cannot differentiate between calls to "#.app1.BB" and "#.app2.BB".

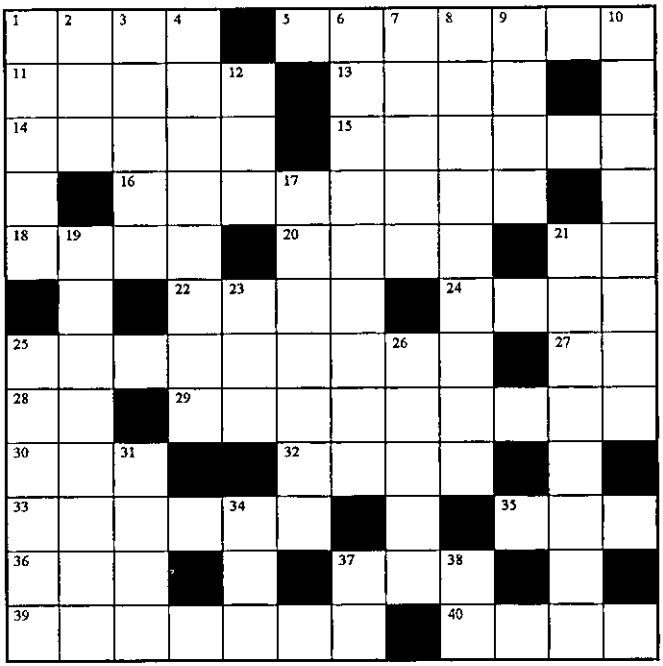
The best solution(s) received will, I hope, be published in a future issue of *Vector* and incorporated into the UCMD code available from www.vector.org.uk.

Solution to Crossword from Vector 15.3



[Vector Production] This faithfully reproduces Gary's original. Vector Production reckons that the 'i-beam' characters are really multiplication signs. What do you think?

Crossword



Across

1. $V[I]\times B[(1+\rho B)-I]$ for I from 1 to ρV, for origin 1 with $(\rho B)=(\rho V)$
5. A random integer from 0 to 4, with 0 half as likely as the others
11. $\iota(\rho V)+(\rho B)$
13. The matrix F with its first $M5$ columns made last
14. The values in A rounded to the nearest integer (.5 down not up)
15. Are all the elements of the vector VN larger than zero?
16. The difference of 102 and one-fifth S
18. $(V\neq 25)/V$
20. Flag the 52s in N
21. Get outta here!
22. 0 where $S=V$, 1 where $S>V$, ¯1 where $S<V$
24. $2\phi'(0\lceil A\div B)\times C'-'ABC\times',\blacktriangledown 0$
25. The index of the first row of the matrix M that CONTAINS the vector V

27. The number of rows and columns in *K*
28. The vector `1 2 3 ...` *L*
29. `(`*RV*`-1),(`*RV*`-2),...,(`*RV*`-(10+`*VC*`))`
30. An empty vector with the same datatype as the scalar *S*
32. Random one-element vector from `¯1` to `¯9`
33. Get objects from a stored workspace IF they're not in the active workspace
35. All but the first *N* elements of the vector *V*
36. The number of dimensions in *A*
37. Deal a card from the deck
39. Famous earless APL programmer, _____ Van Go To
40. Deal five cards from the deck

Down

1. `(⍒`*V*`)[⎕IO]`
2. Is the vector *A* nonempty?
3. Subtract 12 from each element of *V*; then reverse their order
4. Stick onto matrix B a new column, using the numbers in the vector *VR* divided by 20
6. The index of the first row of the matrix *M* that MATCHES the vector *V*
7. Five random numbers from 1 to 25, with replacement
8. `2÷10⍕9÷(9↓`*V*`),9↑`*V* if `(⍴`*V*`)≥9`
9. The available programs
10. The vector *KC* with negative values replaced by zero and padded to length 500
12. A good name for a function that bombs a lot
17. *M*+*Y*-*N* without using +
19. For a bit vector *A*, where *L*←⍴*A*, reshape bits *I* to length *P*, pad to the length of *A*, and compare *A* to see where neither is true
21. `((⍴`*V*`)=0)/`*L*`5`
23. Flag the occurrences of the (start of) substring *S* in the vector *V*
25. The first element of any nonempty array *V*, as a scalar
26. The vector `1 2 3...` up to some random number between 1 and 55
31. Cumulative reduction
34. What the circle ○ function is easy as
37. A random integer scalar from 1 to *T*
38. `⍎⍕⍞'2*5'`

J-ottings 20: A Case of Taken Identity

by Norman Thomson

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

Let's call it `id 3`. Boring? Well, so you might think. What I want to show here is how constructing the identity matrix of order n can lead to an excursion through a wide variety of J's facilities. My intuitive idea of the identity matrix verb is

```
id=: i.=/i.
```

that is, a fork with "equals table" as the pivot verb, following the same transformation "integers" applied both left and right.

If the repetition of `i.` offends anyone to whom economy of expression is one of J's endearing features, use the "reflex" adverb ~ together with either of the conjunctions "atop" or "with"

```
id=: =/~@i.  or  id=: =/~&i.
```

All three of these verbs use six characters. However consult J phrases, and you will find that a solution in four characters is possible:

```
id=: =@i.
```

At first sight this might seem to mean compare `i.n` with itself resulting in `n#1`. However, as often happens in J, appearances are deceptive. = is *not* "equals", it is the monadic verb "self-classify" which provides the "equals table" of the nub of an object with itself. When the object contains no duplicates, this is of course just the same as the equals table of the object with itself. This same table arises from the five-character solution using "raze in" (`e.`) :

```
id=: e.@i.
```

The second shortest of the nine alternatives offered in J phrases is

```
id=: [ D.1@i.
```

and introduces the derivative conjunction `D.` which in general joins a *monadic* verb to an integer `y.` in order to produce its numerical `y.` th order derivative. The verb joined to `D.` in `id` looks like "left" (which implies ignore everything to the right), but is in fact its monadic form "same", corresponding to the function y=x. The first derivative y' of this function is the constant 1, even if the type of the argument is character. If the argument is a vector, each item is regarded as the value of a separate variable, and so the function is, say $y = (x_1, x_2, x_3)$. The result of the derivative conjunction is then the table of all possible partial derivatives of x_i with respect to x_j, which is again the identity matrix.

Digressing for a moment, dyadic `[` is often used in the context of statement separator as in

```
a=.1 [ b=.2
```

Reading the assignments as events which happen incidentally to the main action, the above line is read as `1 [ 2`, returning the left value 1. In a similar way `]` ("right") means ignore the argument on the left. Be careful not to use `]` by mistake as a statement separator; `a=.1 ] b=.2` results in *both* `a` and `b` acquiring the value 2.

On the other hand, the monadic "same" verb in either of its forms `[` and `]` is often used to display the result of a line so that `] a=.1 [ b=.2` and `[ a=.1 [ b=.2` both display the value 1, as well as carrying out the two assignments.

Another variation for `id` in J phrases has the structure of a fork, as is emphasised by the redundant parentheses in the rendering below.

```
id=.(,~) $ ({.&1@>:)
```

The left transformation generates `n,n` from `n`. The right transformation attaches 1 to the `n` fill 0's generated by "take", (thus justifying the subtitle of this article) and so the identity matrix at the head of the article arises from `3 3$1 0 0 0`, or more generally `(n,n)$1,n#0`, an expression with an obvious APL analogue.

A further three variants in the J phrases list require the use of the rank conjunction, since none of the three structural verbs involved "take", "shift" and "shape" are scalar dyadic. The first of these verbs again exploits the fill character of "take".

Ignoring trailing zeros, the rows of say `id 3` are

```
1                   NB. _1 {.1
0 1                 NB. _2 {.1
0 0 1               NB. _3 {.1
```

so what the algorithm does is to generate the series `_1 _2 ... _n` by subtracting `1+i.n` from `i.n`, that is by applying `-@>: to i.n`. Everything is now set up for a fork. The right transformation is simply the verb `1:` providing the constant 1, and the pivot verb of the fork is "take" which must be applied at scalar level, that is at rank 0. Hence the complete verb, again shown with redundant parentheses to enhance readability, is

```
id=:(-@>:@i.) ({."0) 1:
```

The next algorithm is similar but uses "shift" as well as "take" with fill characters to generate the rows

```
1 0 0               NB. 0 |. 3{.1
0 1 0               NB. _1 |. 3{.1
0 0 1               NB. _2 |. 3{.1
```

"Shift" is dyadic and requires two explicit ranks to reflect the fact that the shifts are specified by scalar left arguments to apply to the same rank 1 (vector) right argument.

```
id=:(-@i.) (|."0 1) ({.&1)
```

The next strategy is to generate the zeros *explicitly*, rather than by filling, and then join a 1 on to the end.

```
1                   NB. (0 $ 0) , 1
0 1                 NB. (1 $ 0) , 1
0 0 1               NB. (2 $ 0) , 1
```

Again the rank conjunction is required in order to scalarise the successive left arguments, 0,1 ..n-1 of `$`.

```
id=:(,&1)@($&0)"0@i.
```

Next comes a different implementation of the same strategy. When a verb (e.g. `0&,`) is qualified by an adverb (e.g. `^:`) two separate arguments are required at execution, one by the adverb, and the other by the adverbially qualified verb. For example

```
      (0&,^:3)1
0 0 0 1
```

applies the 0-join 3 times (argument of `^:`) to the starting value 1 (argument of `0&,^:3`). To generate `id 3`, 0 has to be joined to 1 zero, one and two times, as in the table for the immediately preceding algorithm. The adverbial argument is therefore `i.3` and the verb which generates the 1 to which all the joining happens is `1:`. The conjunction between the verbs supplying these two arguments is "tie", and so we have a demonstration of the tie adverb acting in combination with the "power" adverb. The complete form of the verb is

```
id=:0&,^:(i.`1:)
```

Now, you may complain that none of this discussion has taken us one jot further away from the starting point. However in the course of the journey, we have visited forks, "self-classify", "raze in" (`e.`), the derivative conjunction, the "reflex" adverb, the verbs "left" and "right" converging on monadic "same", the non-scalar dyadic verbs "shift", "shape" and "take" leading to the necessary use of the rank conjunction, the conjunctions "atop" and "with", and finally the adverbs "power" and "tie". What algorithm should you choose to generate identity matrices routinely? The choice of course is yours — personally, my vote is for the first one, namely

```
id=:i.=/i.
```

but then I just happen to like circular tours!

Renaissance Data Systems

All APL Books in Print

We are pleased to announce that RDS is now on the web at www.aplbooks.cnchost.com with a display of our entire catalog of APL Books in Print. There are even some classic publications which we have received permission to reproduce. Subjects include APL and Mathematics, Learning APL, and APL for the Professional Programmer. We also carry a few books of interest from the rest of the world!

You may contact us using email at rendata@aplbooks.cnchost.com and place orders at orders@aplbooks.cnchost.com. Credit card payment is available for international customers.

Please note our new mailing address:

Renaissance Data Systems
P. O. Box 313
Newtown, CT 06470.

GENERAL ARTICLES

This section of Vector is intended for general readers who have a working knowledge of APL or J. Authors are encouraged to submit articles which illustrate effective APL applications.

An Approximation of the N-body Gravitation Problem in J

by Andrew Towers
Email: ajtowers@postoffice.newnham.utas.edu.au

Abstract

This paper presents a method of approximating the celestial n-body using the J programming language. The physics involved in this problem are introduced, and some examples given. General code for reading data files is also given.

Introduction

The *problem of two bodies* and the *problem of three or more bodies* are well known problems in the field of Celestial Mechanics. These problems involve projecting the gravitational effects of celestial bodies on each other over a period of time, in order to determine their resulting positions.

In this paper, a method of approximating this calculation through the use of simulation is given, based on the Newtonian law of general gravitation:

> "Two bodies attract each other with a force that is proportional to the mass of each body and is inversely proportional to the square of their distance apart."[1]

Mathematically, this translates to the following formula:

$$F = k\frac{m_1 m_2}{r^2}$$

This formula is herein expressed in the form of J functions, allowing a data file containing values for a number of celestial bodies to be simulated, and the results presented in graphical format.

Structuring the Data

In order to perform a simulation over a number of time units, the data must be structured in an appropriate way. Therefore, the first task was to create an appropriate data structure for the problem. While initially this was going to be a

2-body simulation, it was decided that, in keeping with the array penchant of J, an n-body solution would be more appropriate, and likely as easy to create.

Each of the bodies in the simulation consists of three attributes; a current location in absolute space, an absolute velocity and an absolute mass. For convenience of processing, these are grouped into three arrays; the current position of each body, the current velocity of each body, and the current mass of each body. These three arrays are boxed, forming a single value.

Also for convenience, both the current position of each body and the velocity vector for each body are represented as imaginary numbers. This allows for easy manipulation of data in two-dimensional space. The mass of each velocity is represented as a real number.

During the simulation, a new array of body positions will be generated for each time unit. To accommodate this, the positions form a matrix, each row of which represents the body positions for an instant in time. As the simulation progresses, this matrix is extended by appending the array of new positions.

The initial state of the simulation is read from a data file, the name of which is specified when the simulation is started.

```
5000j5000 5000j5600 5000j3800 7000j2000
0j0 35j0 22j0 _8j19
100000 100 400 1
```

This example shows the format of the data file for the four-body simulation shown later in the paper. The first line represents the initial positions of each of the four bodies, the second line represents their initial velocities, while the final line contains the mass of each body. These values are expressed in arbitrary units.

Breakdown of the Simulation Code

Performing the Simulation

An entire simulation is performed through the use of the *simulate* function. This function is applied to the name of a data file, and results in the display of a graph showing the outcome of the simulation. Note that the graph is a side effect of the function; no result is produced, therefore this is not strictly a function at all.

```
simulate =: display @ (process ^: 200) @ getdata
```

The *getdata* function constructs a boxed list from the data file named by its argument. This boxed list is passed to the *process* function, which will perform a

simulation for one unit of time. This function, raised to power 200, causes its initial result to be re-processed a further 199 times. Finally, the *display* function causes a plot of the resulting 200 sets of points. This arbitrary figure was chosen as it allows a reasonable simulation to be performed without taking an inordinate amount of time.

```
process =: (pos , newpoint) ; newvel ; mass
```

The *process* function is a monadic double-fork. Its single argument, a boxed list representing the current state of the simulation, is passed to each of the three monadic functions. Each of these produces a new value for the appropriate position in the boxed list. These results are subsequently boxed up to form the new state of the simulation. The *pos* and *mass* functions simply un-box the appropriate array out of the boxed argument.

```
newpoint =: last @ pos + vel
```

The new position of each body is the result of summing the current position with the current velocity of each body. As these are both arrays of imaginary numbers and of the same length, it is trivial to perform this calculation. The *last* function isolates the last row of the matrix passed to it, reshaping it into an array. As for *pos* and *mass*, the *vel* function isolates the appropriate array from its boxed argument.

```
newvel =: vel + totalforces % mass
totalforces =: +/ @ forcevecs
```

The new velocity of each body is the most complex calculation. Like process, this function is a monadic double-fork. The new velocity of each body is calculated by adding an acceleration to its velocity, where acceleration is determined by the total effect of all forces acting on a body, divided by its mass. Once again the boxed list is passed to each of the three functions intact.

The total effect of all forces acting on a body can be calculated by summing all the forces acting on that body. The *forcevecs* function takes the boxed list and calculates a matrix of forces between each pair of bodies in the simulation.

```
unitvecs =: lastpos *@-/ lastpos
forcevecs =: forces * unitvecs
```

The *forcevecs* function combines the matrix of unit vectors generated by *unitvecs*, with the matrix of force strengths generated by the *forces* function. Each body has a unit vector of 0j0 in the direction of itself, neatly masking off any effects that a

body may erroneously have on itself as a result of the *forces* calculation. The unit vector between any pair of bodies indicates the direction of the force that the second exerts on the first.

```
ugc =: 6.67 & *
masses =: mass */ mass
forces =: masses ugc @ * scaledists
```

The matrix of force strengths between each pair of bodies is a result of multiplying the mass products of each pair of bodies by the scaled distances between each pair of bodies. Each resulting force is then multiplied by a gravitational constant appropriate to the units in which the input data is expressed. In this case the constant 6.67 has been used, as it relates to the gravitational constant used with real-world gravitation calculations.

```
scale =: % @ *:
distances =: lastpos | @ -/ lastpos
incmask =: lastpos -. @ * @ | @ -/ lastpos
scaledists =: incmask scale @ + distances
```

The matrix of scaled distances is a result of applying the *scale* function to the matrix of distances between bodies. The *scale* function implements the inverse-square law of diminishing effect.

The only catch here is that the *scale* function takes the reciprocal of its argument. As the distance of any body from itself is zero, this will result in a scaled distance of infinity. These unwanted effects would be nicely masked by later multiplication with a vector of length zero, except that zero times infinity yields an indeterminate result.

Because of this, an increment mask is added to the distances matrix before this scaling calculation is applied. The increment mask forms a matrix containing the value 1 in each cell for which the distance between two bodies is zero, and a zero in every other cell. These erroneous calculations will be later masked by zero-length unit vectors.

Reading the Data

The starting state of the simulation is read from a data file, the name of which is the only argument to the *simulation* function. This function makes use of the *getdata* function to read the data file.

```
reshape =: $~ 1 & , @ #
shapepos =: reshape @ pos ; }.
getdata =: shapepos @ readfile
```

The *getdata* function makes use of the *readfile* function, which is a general purpose function for reading a file and boxing the results of executing each line. The *shapepos* function is used to reshape the positions array of the result, to form a matrix of 1 row, and preserving the number of columns.

```
read =: 1!:1@<
dethirteen =: #~ (13 { a.) & ~:
readfile =: split @ dethirteen @ read
```

The foreign conjunction `1!:1` is used to read the textual content of the data file. This is valid in version 3.04d of J for Windows 95. This text is the processed by the *dethirteen* function, which removes all the carriage return characters. This is done because these characters interfere with the execution of lines using the do (`".`) function.

```
prepnl =: (10 { a.) & ,
parse =: < @ ".
cut =: parse ;. _2
split =: }. @ cut @ prepnl
```

The *split* function first prepends the text with a newline character, so as to make use of the cut (`;.`) function of the J language. This function splits the text according to a delimiter, which must be the first character of the text. At each split, the *parse* function is applied to the isolated segment. This function boxes the result of executing the isolated line of text. The *split* function must also behead the result, to remove the empty box that results from the prepended newline.

Displaying the Results

To display the result of the simulation, the J *plot* module is used. This module is provided with version 3.04d of J for Windows 95, and may be unavailable on other platforms or in other versions.

```
display =: 'type dot;backcolor white;title n Bodies' & plot @ preproc
```

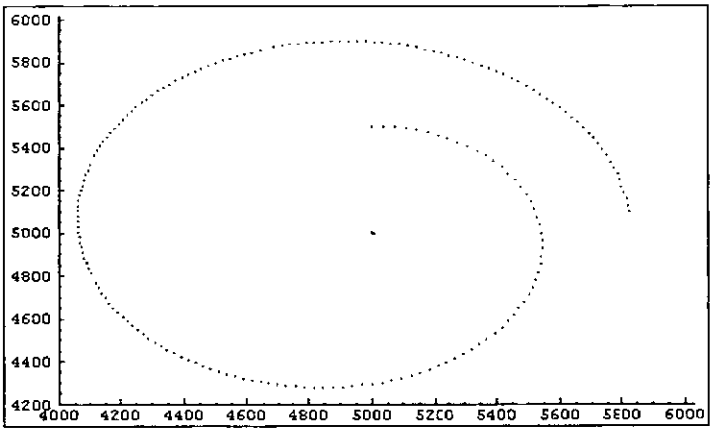
The monadic display function takes as its argument the boxed list of values resulting from the process function. The *plot* function; the interface to the plot module, is used dyadically. Its left argument is a set-up string, specifying the type of graph to be plotted. The right argument, for a dot graph, must consist of a two-box value.

```
xcoords =: {. @ +. @ ,
ycoords =: {: @ +. @ ,
coords =: xcoords ; ycoords
preproc =: coords @ pos
```

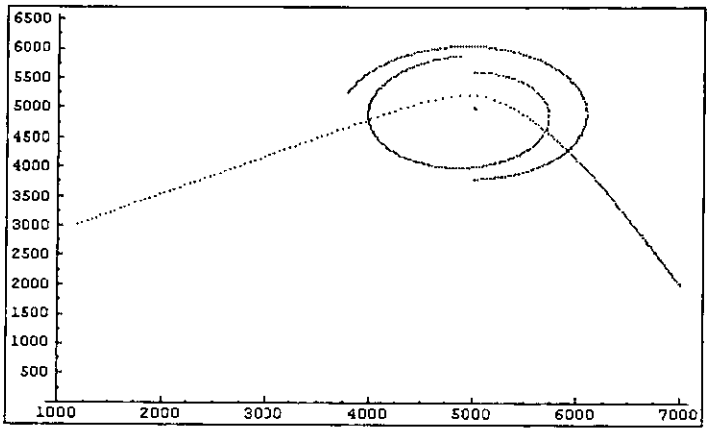
The *preproc* function performs the necessary pre-processing to format the result of the simulation for the *plot* function. A two-box value is constructed; the first box must contain an array of x co-ordinates, the second box an identical length array of y co-ordinates.

Examples

The following graph shows the result of the simulation of a two body problem.



This graph shows a body with relatively small mass orbiting a large mass. As this illustrates, the cumulative inaccuracy introduced by the step-by-step approach prevents a true orbit. Adjustments of initial position, velocity or mass result in either direction result a wider arc.



This graph shows the result of simulation of a four body problem. In this simulation, a second "planet" has been added, and a "comet". The comet shows the sling-shot effect of passing a low-mass body close by a high-mass body at relatively high speed. On approach, the comet accelerates toward the high-mass body. This accelerated velocity enables it to escape the gravitational pull of the high-mass body.

Conclusion

As these examples show, this cumulative approach introduces significant error into the calculations. Alternate methods could be used to calculate the force between objects, resulting in more accurate results.

Reference

[1] Max Born 1965, *Einstein's Theory of Relativity*, Dover Publications Inc., New York.

Appendix 1 - Code

```
NB. fetch the plotting routines
load 'plot'

NB. read in a data file and box up the result of each line
NB. the parse function executes each line using ".
read =: 1!:10<
dethirteen =: #- (13 { a.) & -:
prepnl =: (10 { a.) & ,
parse =: < @ ".
cut =: parse ;. _2
split =: }. @ cut @ prepnl
readfile =: split @ dethirteen @ read

NB. read in the file and reshape the pos array to form a matrix
reshape =: $~ 1 & , @ #
shapepos =: reshape @ pos ; }.
getdata =: shapepos @ readfile

NB. unbox the relevant array
pos =: > @ {.
vel =: > @ {. @ }.
mass =: > @ {. @ }. @ }.

NB. take last row of matrix and ravel
last =: , @ }.~ _1 & + @ #
lastpos =: last @ pos

NB. force of attraction proportional to mass of each body and
NB. inversely proportional to the square of their distance
ugc =: 6.67 & *
masses =: mass */ mass
distances =: lastpos | @ -/ lastpos
incmask =: lastpos -. @ * @ | @ -/ lastpos
scale =: % @ *:
scaledists =: incmask scale @ + distances
forces =: masses ugc @ * scaledists
unitvecs =: lastpos * @ -/ lastpos
forcevecs =: forces * unitvecs
totalforces =: +/"1 @ forcevecs

NB. run the simulation for one time unit
newpoint =: last @ pos + vel
newvel =: vel - totalforces % mass
process =: (pos , newpoint) ; newvel ; mass
```

```
NB. box up the positions as x coords ; y coords
xcoords =: {. @ +. @ ,
ycoords =: {: @ +. @ ,
coords =: xcoords ; ycoords
preproc =: coords @ pos
display =: 'type dot;backcolor white;title n Bodies' & plot @ preproc

NB. display the result of processing the input data
simulate =: display @ (process ^: 200) @ getdata
```

Appendix 2 - Data

Two-body problem

```
5000j5000 5000j5500
0j0 36j0
100000 100
```

Four-body problem

```
5000j5000 5000j5600 5000j3800 7000j2000
0j0 35j0 22j0 _8j19
100000 100 400 1
```

The Manchester Umbrella Decoded

by Prof Oleg Pöhl (olegp@krankenhaus.uni.po)

Acknowledgement

My heartfelt thanks to your editor, Dr Lanzavecchia, for permitting me to share with you this strange voyage of discovery. For many years, I have searched for connections between the forgotten science of ancient Egypt and the many startling discoveries unfolding in the laboratories of today. Never did I think that an obscure souvenir parasol would lead me on a trail which came eventually to link the lost secrets of the Pharoahs with Dolly the sheep, and the arcane notation of a lost computer language.

α - The First Clue

It is many years now since I last dabbled in your language APL, and I had long forgotten the old umbrella which my son gave me on his return from the Manchester convention. Then one morning I read this in the Guardian newspaper, which my son sends me from time to time to keep me warm in the winter:

Alpha of ancient Greece claimed to be Scots symbol of immortality

Maev Kennedy
Heritage Correspondent

IN THE BEGINNING was not Ancient Greece but Stone Age Scotland, according to a retired engineering draughtsman who has spent 15 years peering at marks on rocks.

Edward Peterson is convinced that the symbol Alpha, used by the Greeks around 800BC as the first

Mr Peterson, however, believes the alpha symbol was widely used in Scotland to signify the beginning of a new life of the soul in death. He believes traders from Scotland took a symbol that was already ancient with them to the Mediterranean, where the Greeks took it up.

Mr Peterson, aged 70, from Aberuthven in Perthshire, has published a

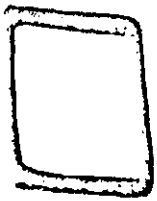
Guardian Newspaper
March 1999

Something stirred deep in my memories, and I began to visualise again the markings on the Manchester parasol. I became quite certain that there were three symbols, repeated four times, and that the first of them could so easily be a crude representation of that early Pictish alpha.

Frantic indeed was my search, and great my jubilation on discovering again my lost treasure, which not only confirmed my initial conjecture but would quickly lead me to even greater riches.

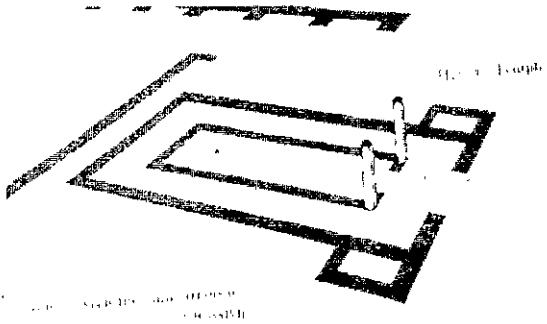
Here we have again that Manchester alpha, re-photographed and image-enhanced by the finest modern technology, but still retaining that inexplicable sacredness of an 8,000 year old totem. It was adopted by the Nazarenes as their 'fish' symbol some 50 years before Christ was born. Indeed, "Nasrani" remains in use today as the Arabic word for a shoal of small fish as well as for 'Christian'. In the New Testament it occurs many times, as in Ιησουν τον Ναζωραιον [John 18,5] and has been in constant use by Christians since the persecutions of ancient Rome.

The Second Clue



I believe that this is known to your readers as {quad}, but in spite of the unfortunate distortion caused by stretching of the fabric, it is clearly a plan of Solomon's Temple, central to the mysterious discoveries of the Knights Templar and by implication to all the strange rites of modern freemasonry.

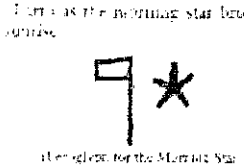
Compare it for a moment with the known layout of the Temple, as copied precisely in the construction of Rosslyn chapel in the mid fifteenth century:



Plan from "The Hiram Key"
Knight & Lomas
Arrow, 1997
page 423

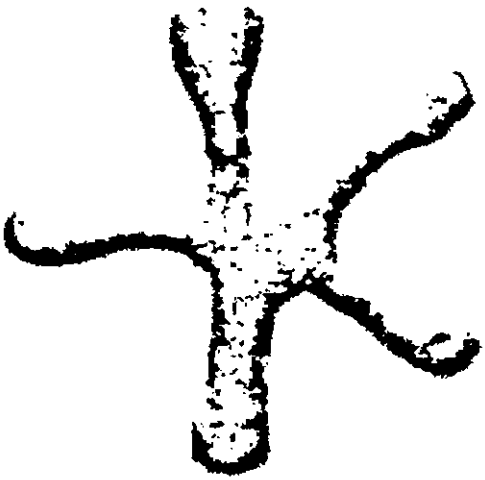
The proportions of the APL □ match the proportions of the Temple so perfectly that this surely can be no co-incidence? We shall return to Rosslyn in a moment, for it is here that the strangest part of the mystery unfolds. But first to the central, third symbol on the parasol:

Power and the Dawn Star



Now I *knew* that I had stumbled on something truly mind-shattering. The Sumerian symbol for the 'dawn star' is a FIVE POINTED star - used by the Egyptians from around 4000BC to symbolise the rebirth of the King as Osiris and his journey to the stars.

This is one of the most ancient symbols known to man, having come to Sumer from the ancient civilisations which were drowned by the great catastrophe sometime between 10,000 and 8,000BC [*Heaven's Mirror*, Graham Hancock, 1998]. The computer languages of the world use only one symbol - the six-pointed star - for multiplication, and yet here on the Manchester Umbrella I find:



The revelation came upon me like a great light breaking out in a darkened room - your APL language alone uses the 5-pointed star as the symbol not for multiply, but for 'Power'. Surely this also can be no co-incidence?

From here it a is short step to note that the 'triple tau' (King, Priest, Prophet) can easily be rendered in APL as Lefttack+Righttack+Downtack, and that one of the only symbols allowed in a variable name other than A-Z is Δ - the pyramid - the symbol traditionally employed by APL programmers as the preface to their most sacred (global) variables.

The Link with Freemasonry

I quickly realised that there existed many more odd 'co-incidences', for example the original statement separator {diamond} was formed by overstriking ∧ and ∨ to produce something identical to the badge of freemasonry (the square and compasses). So - why was {diamond} never implemented by mainframe APLs on IBM machines ... surely this can only have been *because it was considered too sacred to be used.*

The Final Secret Revealed

I communicated these ideas to my son, and within days I received the following communication by electronic mail:

```
The likely repository of the secrets of the last true King of Egypt is the
'chapel' of Rosslyn in Scotland. This is where they were secreted after the
Templar William St Clair hid the excavated remains of Solomon's temple back in
1370 after the Templars were banned by the Catholic church. So far, so
uninteresting.

What else just happened recently near Rosslyn??  That's right .... DOLLY THE
SHEEP.

Simple really - the kings of Egypt attained everlasting life by cloning. At
the death of the old king, the new king was duly drugged up to the eyeballs,
his sperm collected (means uncertain) and used to start the next king. His
sister was naturally used to carry the child, giving rise to incorrect stories
of incest. Actually it was just a simple clone job - the sister was necessary
to avoid problems of rejection of the implanted foetus.

So ... the secret is out. The method was carried out of Egypt by Moses, holed
up in the Ark of the covenant for a few thousand years, rescued by the
Templars, hidden in Rosslyn Chapel and finally revealed to the Ministry of
Agriculture who used it with immediate effect to create Dolly. Oddly enough,
the experiment has not yet been successfully repeated!
```

The final proof (as if any were needed) ...

The usurpers who ran the lower kingdom of Egypt in the 'First Intermediate Period' were known as "The Shepherd Kings" and the sacred words which are told to a master of freemasonry on initiation are ....

"Ma'at-neb-men-aa Ma'at-ba-aa"

Prof Oleg Pöhl
1st April 1999

Prof. Pöhl may be contacted via Vector production [Ed]

The Investment Models of a Finnish Pension Company

by Antero Ranne (antero.ranne@ilmarinen.fi)

The Investment Models

The Mutual Pension Insurance Company Ilmarinen is one of the companies managing the statutory employment pension in Finland. The investment models for actuarial use have been developed in Ilmarinen starting from the year 1993. The main parts are:

- financial time series data management
- time series analysis programs
- investment simulation models
- portfolio optimization models.

The models have been used e.g. for the following purposes:

- Developing the statutory solvency requirements for the Finnish pension companies and other insurance companies
- Developing new rules for determining the technical interest rate of Finnish pension companies
- Analysing the risks connected with the investment strategy of the company.

Financial Timeseries Data

The models are based on financial time series from 12 countries. These include e.g. inflation, GNP growth rate, interest rates, share price indices, dividend yields etc. The longest of the series start from the year 1950. The data are saved in an three-dimensional APL array. There are programs for adding new data and for output as tables or figures.

Timeseries Analysis Programs

Various time series equations are defined for the variables in the investment model. Programs exist in APL to estimate the parameters of these equations. Although commercial statistical programs can be used for more thorough

analyses, it is convenient to be able to make calculations in the same environment where the data are.

As an example it is shown how the parameters of the inflation model are estimated. The inflation rate may be influenced by external shocks which in the model are represented by oil price movements. The inflation $i(t)$ in year t may be modelled e.g. by the equation

$$i(t) = m + a_1\,[i(t-1)-m] + a_2\,[i(t-2)-m] + b_0\,o(t) + b_1\,o(t-1) + e(t)$$

where $o(t)$ is the oil price inflation and $e(t)$ are the residuals or the stochastic part of the model. The parameter values are obtained by iteration. For the intermediate parameter values m, a_1, a_2, b_1 and b_2 we calculate the residuals $e(t)$, as well as the partial derivatives

$$\frac{\partial e(t)}{\partial m} = -1 + a_1 + a_2, \quad \frac{\partial e(t)}{\partial a_1} = -x(t-1) + m, \quad \text{etc.}$$

By ordinary linear regression we find the coefficients d_1, ... d_5 in the equation

$$e(t) = d_1 \frac{\partial e(t)}{\partial m} + d_2 \frac{\partial e(t)}{\partial a_1} + d_3 \frac{\partial e(t)}{\partial a_2} + d_4 \frac{\partial e(t)}{\partial b_0} + d_5 \frac{\partial e(t)}{\partial b_1}\,.$$

The new parameter values are calculated by subtracting the d-coefficients:

$$m' = m - d_1\,, \ldots\,, b_2' = b_2 - d_5\,.$$

The solution is easy in APL because the linear regression part can be calculated directly using ⌹, as is shown in the following programs. The variable *"inf"* is the inflation and *"oil"* the oil price inflation.

```
par←Infparam;p2;e;d;c
p2←1+par←5⍴0
:while 1E¯3<⌈/|par-p2
  ⎕←p2←par
  e←p2 Resid inf,[1.1]oil
  d←p2 Deriv inf,[1.1]oil
  c←e⌹d
  par←p2-c
:endwhile
```

```
e←par Resid vars;x;y;m;a1;a2;b0;b1;i
(x y)←⊂[1]vars
(m a1 a2 b0 b1)←par
x←0 0,x-m ⋄ y←0 0,y ⋄ e←(⍴x)⍴0
:for i :in 2↓⍳⍴e
  e[i]←x[i]+(-a1×x[i-1])+(-a2×x[i-2])+(-b0×y[i])+(-b1×y[i-1])
:endfor
e←2↓e

d←par Deriv vars;x;y;m;a1;a2;b0;b1
(x y)←⊂[1]vars
(m a1 a2 b0 b1)←par
d←((⍴x),1)⍴¯1+a1+a2
d←d,-¯1↓0,x-m
d←d,-¯2↓0 0,x-m
d←d,-y
d←d,-¯1↓0,y
```

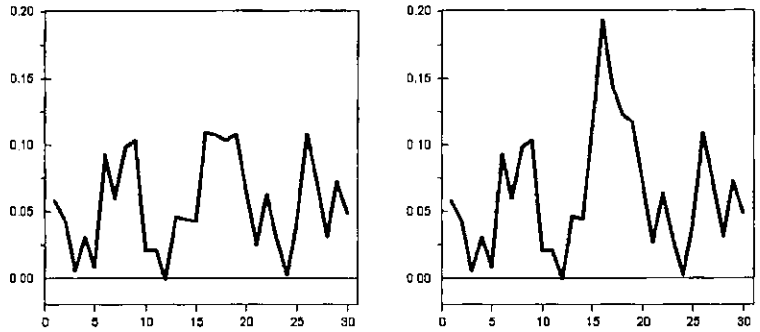
The calculation takes only a few steps. The same principles can be used for other quite complicated equations.

Investment Simulation Models

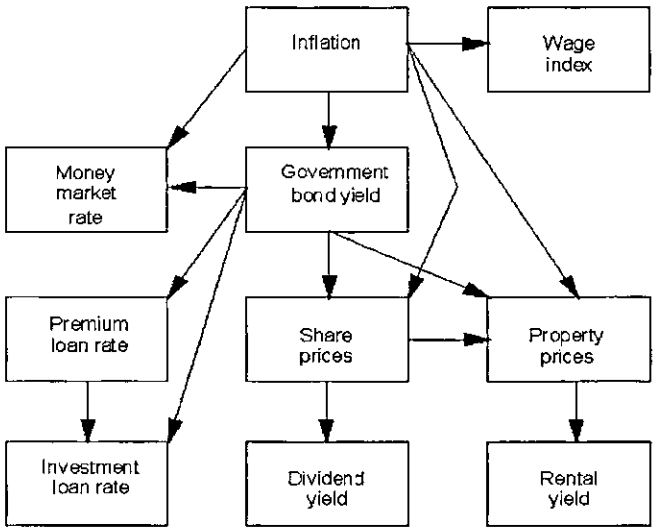
When the parameter values have been established, the time series equations can be used to simulate the variables. The residuals are then generated as random numbers using the Wilson-Hilferty algorithm (see e.g. Daykin *et al.*, 1994). The numbers are calculated using the following program:

```
 Z←V Random N;M;S;A;B;C
⍝ N random numbers from distribution with
⍝ mean = 0, std deviation = 1 and skewness = V
⍝ first numbers from normal distribution
 S←(÷1+M)×?(N,2)⍴M←2147483646
 Z←(2oo2×S[;2])×(¯2×⍟S[;1])*0.5
 →(0=V)/0
⍝ skewness is added by Wilson-Hilferty-algorithm
 (A B C)←((V*2)÷108),((V÷6)-6÷V),2÷V
 Z←(A×(Z-B)*3)-C
```

Figure 1 shows an example of simulated inflation numbers using the parameters obtained by the programs shown above. The second version contains in year 15 an inflation shock generated by the oil price variable.

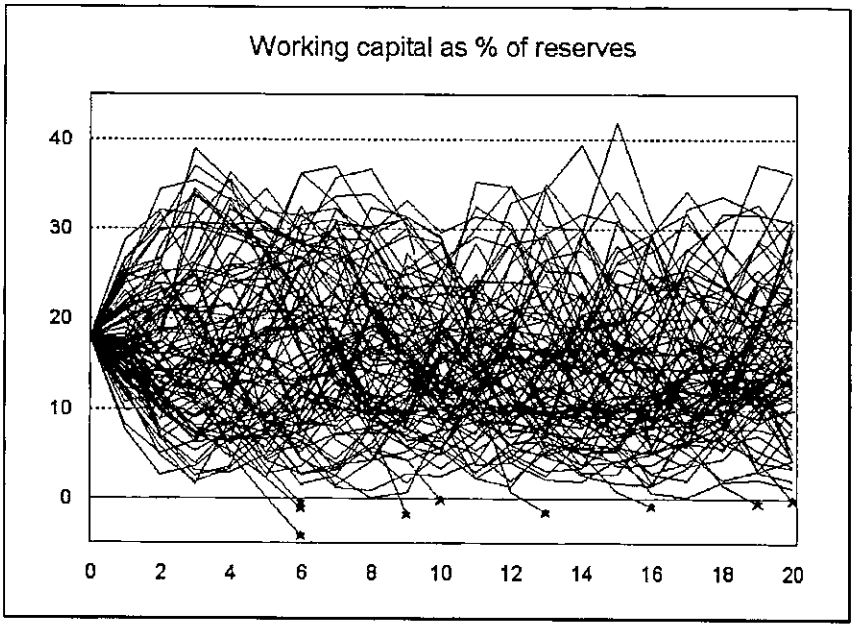


The structure of the whole investment model is shown in Figure 2. A detailed description can be found in Ranne (1998).



The purpose of the model is to generate investment yields that have realistic means, standard deviations and correlations in the long term. These are used to estimate the investment risks of a company.

The investment variables are combined with a forecast describing the development of the whole pension insurance company and various output variables are calculated (e.g. premiums, pension expenditure, reserves, bonuses, profits or losses). The most important output variable is the working capital, which is the difference between the total assets and the liabilities of the company and which acts as a buffer against variations in the investment results. The following picture shows an example of simulated working capital as per cent of the reserves.



A thousand simulations for 20 years takes about 2 minutes 40 seconds on the present computer (300 MHz Pentium, 128MB memory).

Optimization Models

In the optimization problem there are different investment categories (e.g. bonds, shares, property, cash). The investment returns have means μ_i , standard deviations σ_i and correlation coefficients ρ_{ij} . Standard deviation is used to measure the risk level of the investments.

The proportion invested in category i is x_i ($\Sigma_i x_i = 1$). The mean μ and standard deviation σ of the whole portfolio is

$$\mu = \sum_i \mu_i x_i$$

$$\sigma = \sqrt{\sum_{i,j} c_{ij} x_i x_j}$$

where c_{ij} is the covariance $c_{ij} = \sigma_i \sigma_j \rho_{ij}$.

The problem is to find the portfolio with the lowest risk σ corresponding to a given mean return μ. (Alternatively one could find the maximum return corresponding to a given risk.) In practice there are often also minimum and maximum proportions for different investment categories.

Mathematically the problem is therefore formulated as

minimize the function $\Sigma_{i,j} c_{ij} x_i x_j$

using conditions $\Sigma_i x_i = 1$
$\Sigma_i \mu_i x_i = \mu$
$m_k \le x_k \le M_k, \ k = 1,...,n$.

By standard mathematical methods it can be shown that the solution may be found by solving the following group of equations:

$$2 \Sigma_i c_{ik} x_i + \alpha + \beta \mu_k = \lambda_k, \ k = 1,...,n$$
$$\Sigma_i x_i - 1 = 0$$
$$\Sigma_i \mu_i x_i - \mu = 0$$

where in each investment category i one of the following conditions is true:

a) $x_i = m_k$
b) $x_i = M_i$
c) $\lambda_i = 0$.

For each combination of these conditions the resulting system of equations is linear, and so it can be solved in APL using the function ⌹.

One has to solve the equations for different combinations of the conditions a–c. The solution that fulfils all the conditions and has the smallest variance is finally chosen as the answer to the optimization problem.

The part of the program solving the linear equations is shown in the following. The program uses the global variables:

Mu	vector	mean values for investment returns
Cov	matrix	covariances
Min	vector	minimum values
Max	vector	maximum values.

Since it is possible that a given system of equations has no solutions, there is a matrix division function that returns the empty vector instead of stopping in domain error:

```
Z←X MatDiv Y;⎕elx
⍝ Matrix division
⍝ Result is empty, if no solution
⎕elx←'Z←⍬ ◇ →0'
Z←X⌹Y
```

The second function finds the distribution for a given mean return *"mean"* and conditions that are based on the argument *"list"*:

```
 z←list Optimal mean;n;N;MAT;C;k;i
⍝ Finds the optimal distribution for given mean return "mean"
⍝ Investment return means in "Mu", covariances in "Cov",
⍝ maximum and minimum proportions in "Max" and "Min"
⍝ list[k]=0: no restriction in investment category k
⍝ list[k]=1: use minimum for category k
⍝ list[k]=2: use maximum for category k
 N←2+2×n←⍴Mu
 MAT←(2⍴N)⍴0
 MAT[⍳n;⍳n]←2×Cov
 MAT[⍳n;n+1]←MAT[n+1;⍳n]←1
 MAT[⍳n;n+2]←MAT[n+2;⍳n]←Mu
 MAT[⍳n;n+2+⍳n]←-(⍳n)∘.=⍳n
 C←(n⍴0),1,mean,n⍴0
 :for k :in ⍳n
  i←n+2+k
  :select list[k]
  :case 0
    MAT[i;]←i=⍳N
    C[i]←0
```

```
    :case 1
      MAT[i;]←k=⍳N
      C[i]←Min[k]
    :case 2
      MAT[i;]←k=⍳N
      C[i]←Max[k]
    :endselect
  :endfor
  z←n↑C MatDiv MAT
```

One has to execute the program for different values of the variable *"list"* and find the solution that fulfils all the conditions and has the smallest variation. The part of the program doing this selection is not shown here.

References

Daykin, Pentikäinen, Pesonen: *Practical Risk theory for Actuaries*, Chapman and Hall, London, 1994

Ranne, A.: *The Finnish Stochastic Investment Model*, Transactions of the 26th International Congress of Actuaries, Vol. 7, Birmingham, 1998.

The Subset Sum Problem

by Alan J. Wilson (Awilson@hrfs.co.uk)

The following is a good example of the awesome problem solving power of APL.

The Problem

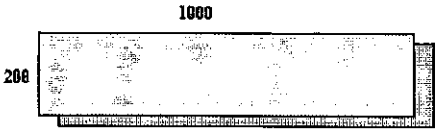
Suppose you have 200 positive integers and want to find out quickly whether any subset of them adds up to 1000.

The Solution

We create an array of zeros of size 200 by 1000.

We enter the first of the 200 numbers in the first row in the column corresponding to its size, e.g. if it was 23 we would put 1 in row 1 column 23.

We then take a second number and displace the original array as follows:



The displacement is one down and then across by the amount of the second number. We chop off the overhang which is shaded in, pad out the top row and left hand gap with zeros and add together the displaced row with its undisplaced self. APL does this in one line of code, see line 5 of *CREATE_MAT* below.

Finally we enter a 1 in the top row in the column in a position corresponding to the size of the second number. The total of the entries in the array is now 3 (unless some have dropped off the edge).

We repeat this process for each number, the total number of entries after the n^{th} number is processed is a maximum of $2^n - 1$. If a particular number is greater than 1000 we reject it, see line 2 of *CREATE_MAT*.

We continue this process until either we have an entry in column 1000 or we have used all the numbers up. The function listed in fact processes all the numbers and

gives distributions of all the totals from all possible subsets, subject to a limit of the array size.

An entry in column 1000 implies a subset of the numbers used to date adds up to 1000. We can determine when the entry first appeared, hence the last number used is in the required subset. We can use this to establish the last of a set of numbers which add up to a reduced target by solving the problem with target 1000 - the last number. By repeating the process we finally obtain the required set of numbers.

A useful test of the function is to use the numbers 1 to 49. The function produces in the sixth row of M totals of 21 to 279, when run as follows:

```
      (⍳49)  CREATE_MAT  6,279
```

The total of the numbers in the row is 13,983,816 which is the number of ways we can select 6 numbers from 49, i.e. `6!49`. `M[6;]` is the distribution of totals of all possible sets of lottery numbers.

Many types of dynamic programming problems can be solved by this technique and variations of it.

```
     ∇ lhs CREATE_MAT  rhs ;N
[1]    M←rhs⍴0  ⋄ N←1
[2]    lhs←lhs[(lhs≤rhs[2])/⍳⍴lhs]
[3]    M[1;lhs[1]]←1
[4]        :while (⍴lhs)≥N←N+1
[5]          M←M+rhs↑0,[1]((rhs[1],lhs[N])⍴0),M
[6]          M[1;lhs[N]]←1+M[1;lhs[N]]
[7]        :endwhile
     ∇
```

TECHNICAL SECTION

Contents

OpenGL Workspace
&
Russian Student Fund

Under the inspiration and guidance of Alexander Skomorokhov, Alexei Zalivine, a Russian Information Processing student in Obninsk, has written about 150 functions to access the OpenGL API functions under Microsoft Windows using Dyalog APL/W version 8 (32 bit). Included in the workspace are a number of utility functions to facilitate the use of the APIs as well as an interactive tool to experiment with the graphic image. For a more detailed description, please see the April, 1998 issue of *Vector*, pages 128-142.

To make it possible for Alexei and some of his colleagues to attend APL99, a Russian Student Fund has been established. For $99 you will receive a copy of the software and the Vector article. $50 will be contributed to the Fund and the remainder will compensate Alexei for his work (capitalism, you know) and cover the distribution costs. For $150, the book, *OpenGL Superbible*, by Wright and Sweet, will be included as well. Additional contributions to the Fund are, of course, most welcome.

Please send a check (preferred in US) or money order in $US made out to Renaissance Data Systems, or credit card information to: Renaissance Data Systems; P.O. Box 421; Georgetown, CT 06829

Additional information may be posted to our web site, www.aplbooks.cnchost.com. Additional information on OpenGL may be found at www.opengl.org.

Hackers' Corner: Don't Clip my Children!

by Adrian Smith (adrian@cgsl.demon.co.uk)

Motivation

"The great storm of computer progress moves on, leaving behind many puddles, in which team interesting life ..." (Ted Nelson, Stanford, 1991)

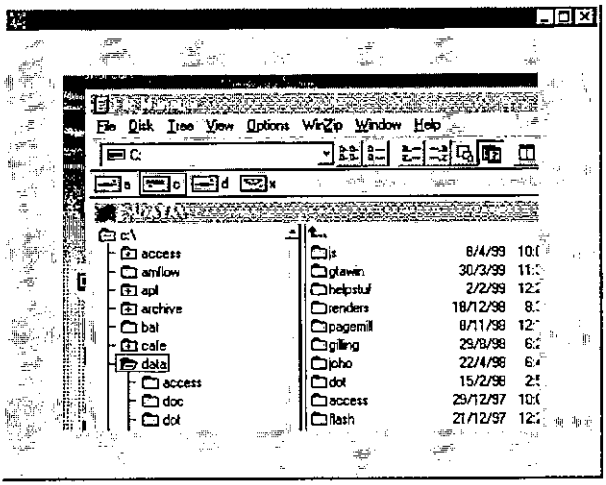
Somewhere between October 1998 and January 1999 the great storm of computer progress swept through Basingstoke, leaving behind it a subtle change to the Dyadic interpreter which few people noticed, but which had some rather bizarre consequences for one particular application which just happened to be dominating my life at the time. It also upset the children, whose 'Mystro' adventure puzzle game developed strange rectangular 'windows' to the desktop (or File Manager or whatever else was underneath at the time) – something funny was going on, and I was going to have to find out what it was!

Investigation

Obviously the first thing to do was to find out what I was doing to trigger this strange behaviour, and then to make up a sample function to illustrate the problem. It turned out that my problem was caused by exploiting a very useful (undocumented) feature of the interpreter which allows you to set a subform as `('BCol' 0)` – this completely prevents it from painting, but it still registers mouse events, and it has a 'Cursor' property. This is exactly what you need if you have a pictorial adventure game where various areas on the picture are marked as 'hot spots' which either jump to another picture, or run some kind of challenge where you need to type in the right code to proceed. More immediately relevant, it lets you simulate 'active desktop' with illuminating shortcuts where the icon labels magically change on mouse-over.

Here is the sample function, and if you turn the page, you can see the weird effect it produces ...

```
     ∇goo
[1]   'ff'⎕WC'form'
[2]   'ff.sub'⎕WC'subform' ''(10 10)(80 80)('bcol' 0)
     ∇
```



The problem is that not only has Dyalog (correctly) left the subform unpainted, but it has carefully avoided painting the underlying form too! After several conversations with the Basingstoke helpline it turned out that they had added a 'ClipChildren' style to the 'Form' object, apparently to reduce flicker when Windows98 or NT users set 'Show contents while dragging' in Display Properties. This also explained another effect I had become aware of – forms with lots of child controls were being drawn with holes cut out under the controls, which were then drawn over the holes. Running on a black desktop with a (fairly) slow graphics card, you really notice this.

The Solution

```
     ∇setwdw obj;gwl;swl;hdl;bits
[1]   ⍝ Zap the 'Clipchildren' style on parent object <obj>
[2]    'gwl'⎕NA'U user32.C32|GetWindowLongA I I'
[3]    'swl'⎕NA'U user32.C32|SetWindowLongA I I U' ¯
[4]    hdl←obj ⎕WG'HANDLE'
[5]    ⎕←bits←(32⍴2)⊤gwl hdl ¯16
[6]    (7⊃bits)←0
[7]    ⍎'swl hdl ¯16(2⊥bits)'
[8]    ⎕←bits←(32⍴2)⊤gwl hdl ¯16  ⍝ Debugging code - remove
     ∇
```

The 7th bit is the one that you need to zap to reverse the change and get back to the old settings. Zapping various of the other bits has bizarre and unpredictable effects, and is not recommended. Simply run this function, passing it the name of your main form after creation and your children will no longer be clipped.

```
     ∇goo
[1]   'ff'⎕WC'form'
[2]    setwdw'ff'
[3]   'ff.sub'⎕WC'subform' ''(10 10)(80 80)('bcol' 0)
     ∇
```

TECHNICAL CORRESPONDENCE

Property Names in APL+Win

From: David Siegel March 1999

In Vector Vol.15 No.3, Jonathan Barman (in his article "VideoSoft FlexGrid Pro with APL+Win") stated:

> "APL+Win requires that you prefix any OCX control properties with an 'x', methods with an 'X', and events with an 'onX'."

This is incorrect. For both OCX and COM interfaces, the properties are exported both with AND without the prepended 'x', 'X', or 'onX'. The only exception is where an OCX control (or COM object) has a method, property, or event whose name duplicates one of the standard APL properties, methods, or events. For example, "size" is a standard APL property. If an OCX control exposes a 'size' property to which you wish direct access, you must use 'xsize' because a reference to 'size' will refer to the APL standard property (a cover for a Windows standard interface). However, a property such as 'FixedCols' may be accessed either as 'FixedCols' or as 'xFixedCols' since this is not the name of a standard APL property. Use of the latter guards against the (rather unlikely) possibility of a standard 'FixedCols' property being added to all APL+Win objects in some future release, of course.

This is a minor point, but it was mentioned as a difference between the Dyalog APL and APL+WIN interfaces. The actual difference is that as Dyalog exports all names without a prefix, access to the standard properties is lost if the control defines a property with the same name, whereas APL+Win permits access to either the standard property, or to the control-specific property in such cases.

David E. Siegel Software Developer, LEX2000 Inc (a COGNOS affiliate)
[Associated with APL2000]

I have no doubt that this is true, but the only help I could find was in Refv3c04.doc in the paragraph headed "Properties of an OCX Control" where it states: You can use the properties of an OCX control in APL GUI by prefixing their names with lowercase letter

x. Similar statements about the X and onX are made in the "OCX Methods and Events" paragraph. I cannot find the fuller explanation given by David Siegel anywhere in the manuals.

It seems much safer to include the leading x X or onX in case a new property, method or event is introduced into APL+WIN which might mask the OCX property method or event. Jonathan

---

Iterated Function Systems

From: Norman Thomson

I read the Reiters' article on the above topic (Vector vol. 15 no.2 pp.104-120) with great pleasure. However I would like to suggest that greater advantage could have been taken of the J interpreter in using complex numbers to generate rotation matrices, and "amend" to tidy up the programming of homogeneous coordinates. To start with, use verbs `id` and `di` to define an identity matrix and the coordinates of its diagonal:

```
   id=: i.=/i.  NB. identity matrix of order y.
   di=: <"1@(i.,.i.)      NB. coordinates of diagonal elements of id y.
   id 3
1 0 0
0 1 0
0 0 1
   di 3
```

0 0	1 1	2 2

Now use the properties of "amend" to make scaling matrices from `id 3`:

```
  scale=: 3 : 'y.(di 2)}(id 3)' NB. scaling matrix y.= x,y scale factors
  scale 0.1 0.5
0.1 0 0
  0 0.5 0
  0 0 1
```

A verb to generate a translation uses a similar technique to generate the indices of the first two items of the third row:

```
trans=: 3 : '(y.,1)2}id 3'   NB. translation matrix y.= x,y shifts
```

```
   trans 0.4 0.6
    1   0 0
    0   1 0
  0.4 0.6 1
```

The rotation matrix is generated by using Euler's formula, followed by +. which generates a 2-item vector from the real and imaginary parts

```
   cs=:+.@^@j. NB. make vector (cos y. , sin y.) from exp(iy.)
```

Now use a gerund, which the Reiters extol in order to generate the 2x2 matrix (c -s, s c):

```
   rot2=: cs`(|.&cs&-)`: 0  NB. 2 dimensional. rotation matrix, y.=angle
   pi=:1p1

   rot2 pi%3
        0.5 _0.8660254
  0.8660254         0.5
```

(As an aside, the result of rotating a single point with coordinates (x,y) through an angle a is given by xjy*j.a and so the hook r2=:*^@j. describes such a rotation, that is

```
   1j6 r2 1
  _4.50852j4.08328
```

produces a result equivalent to

```
   (rot2 1)+/ .*1 6
  _4.50852 4.08328                    )
```

Finally transform to homogeneous coordinates using amend as before:

```
   rot=:3 :'((rot2 y.),.0)(0 1)}id 3'  NB. rotn.matrix for hom. coords, y.=angle
   rot pi%3
        0.5 _0.8660254 0
  0.8660254        0.5 0
          0          0 1
```

The matrices d, e, and f on p.113 are then simply defined as

```
   d=:scale 0.97 0.97
   e=:rot -2r19*pi
   f=:trans 0.189 _0.14
```

The seven lines with comments collectively make up a little package for these transformations.

Working with Formula One in APL+Win

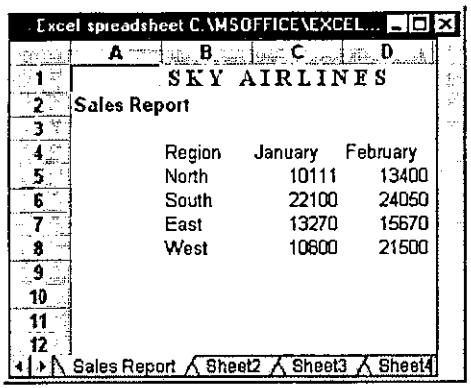
From: Ajay Askoolum 18th January 1999

Very interesting to read Jonathan Barman's *VideoSoft FlexGrid Pro with APL+Win* in Vector 15.3. I have been using another OCX control, Formula One, from Tidestone Technologies. Formula One offers two nice features

1. It can read Excel spreadsheets (without graphs), tab delimited text files etc.

   For example,

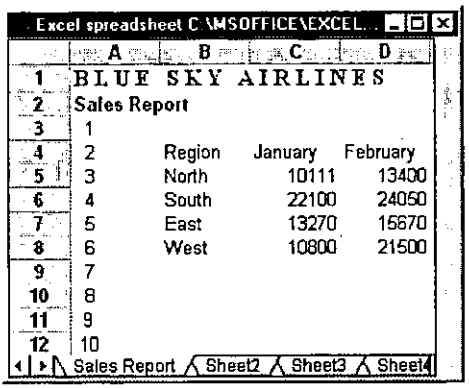
```
ReadExcel 'C:\MSOFFICE\EXCEL\EXAMPLES\SALES.XLS'
```




```
∇ ReadExcel R
[1]   ⍝ Demo FORMULA ONE reading EXCEL spreadsheets
[2]
[3]   'EXCEL' ⎕wi 'Delete'
[4]
[5]       ⎕wself←'EXCEL' ⎕wi 'New' 'Form' 'Hide'
[6]           ⎕wi 'caption' ('Excel spreadsheet ',R)
[7]           ⎕wi 'Set' ('where' 0 0) ('size' 15 60)
[8]
[9]       ⎕wself←'EXCEL.SS' ⎕wi 'New' 'VC Formula One 5.0 Workbook'
[10]          ⎕wi 'Set' ('where' 0 0) ('size' 15 58)
[11]          ⎕wi 'XHeapMin'
[12]          0 0⍴⎕wi 'XRead' R 4
[13]          ⍝ 2=Excel 4, 3=TAB delimited text files 4=Excel 5.0 etc
[14]
[15]
[16]  'EXCEL' ⎕wi 'Wait'
      ∇
```

2. It has an array facility with its properties (note the []).

```
'EXCEL.SS' ⎕wi 'xTextSRC[]' (1,(,['']2+⍳10),1) (⍪,['']⍳10)
```



However, you won't find the array extension documented with the Formula One package. Most cell manipulation properties support it.

In general, in order to find all the properties, methods and events of an OCX supported by APL+Win, create an instance of it (EXCEL.SS, here), then

```
methods←⊃'EXCEL.SS' ⎕wi 'methods'
properties←⊃'EXCEL.SS' ⎕wi 'properties'
events←⊃'EXCEL.SS' ⎕wi 'events'
```

Having established the methods, properties and events, basic documentation on any one of these can be displayed in the APL session. For example,

```
'EXCEL.SS' ⎕wi '?xTextSRC[]' ⍝ Note single ?

xTextSRC      Sets or returns the text value of the specified cell.
  Ref         String <- xTextSRC(Long nSheet, Long nRow, Long nCol)
  Set         xTextSRC(Long nSheet, Long nRow, Long nCol) <- String
```

or

```
'EXCEL.SS' ⎕wi '??xTextSRC[]' ⍝ Note double ?
```

which displays the text above in the APL session and call the Formula One help file and display the topic *xTextSRC[]*. However, the help file does not contain the array topics. Therefore you'd have to look for TextSRC.

This way of finding the properties, methods and events of any OCX works. First create an instance of the OCX, in this example Excel 5.0

```
      ∇ Excel5
[1]    '#' ⎕wi 'Reset'
[2]    'XL' ⎕wi 'New' 'Excel.Application'
[3]    'XL' ⎕wi 'caption' 'APL running EXCEL'
[4]    'XL' ⎕wi 'visible' 1
[5]
[6]    'Now Open a workbook'
      ∇
```

This instance of Excel 5.0 does not open a workbook, so it must be done manually. Excel 97 opens a workbook. Now, create some children

```
      ∇ ExcelHierarchy
[1]
[2]    ⍝ Workbook must be open
[3]    'XL' ⎕wi 'Workbooks>XL.WB' 1
[4]    'XL.WB' ⎕wi 'xActiveSheet>XL.SH'
[5]    'XL.SH' ⎕wi 'xUsedRange>XL.UR'
[6]    'XL.UR' ⎕wi 'FormulaR1C1'
[7]    'XL.UR' ⎕wi 'Formula'
[8]    'XL.UR' ⎕wi 'ClearContents'
      ∇
```

The properties, methods, events of each object can be found in a similar way :

```
properties←⊃object ⎕wi 'properties'

Try

'XL.SH' ⎕wi 'properties'
'XL.UR' ⎕wi 'properties'
```

Finally, what OCXs are available to APL+Win? That is, which OCXs are registered? On my PC,

```
⍴'#' ⎕wi 'XInfo'
466 3
```

You can filter the OCXs

```
⍴'#' ⎕wi 'XInfo' 'Excel'
4 3
```

The last expression returned all the OCXs with the word Excel in their registration.

OCX vendors simply do not mention APL in their documentation. However, the Formula One web site occasionally documents APL/FORMULA ONE problem resolution.

A Technical Note on Counting Moves for Generalized Towers of Hanoi

from Norman Thomson

I enjoyed Howard Peelle's article in Vector Vol.15 No.3 pp.38-46, not least because of its accustomed clarity. Its contents prompt me to make a number of observations.

First, why is there so much about *triangles* which are structures foreign to J? If Pascal had had J available (and just think how he would have loved it!) I surmise that we would never have heard of his triangle, but rather of his *table*, defined by `Pt=:! ./~@i.`) for a given size `y.` .

Page 43 contains the final version of an algorithm for counting the number of generalized Hanoi moves. In this, the phrase `(+/ .* Powers)` is used to multiply a vector by successive powers of 2. This can be more readily accomplished by `#.&|.` thereby making the verb `Powers` redundant, and incidentally speeding up this part of the algorithm by a factor of two.

On p.45 Howard notes the relation between the Pascal table and the Hanoi table on p.44. Having developed M, this relationship can be demonstrated by an alternative way of generating `Hanoi` :

```
(2+i.11)M each each ft 11     NB. each=:&>
```

To summarise some of the results which we are encouraged to obtain from the `Triangle` verb

```
+/(1 1,:0 0)&Triangle Repeat m   and   +/(1 1,:0 1)&Triangle Repeat m
```

give the final columns (bar the initial items) of `Pt m+1` and `Pt m+2` ;

```
+/"1(x 1,:0 0)&Triangle Repeat m  gives the powers of x+1, and
+/"1(x 1,:0 1)&Triangle Repeat m  gives the scans of the powers of x+1
```

`x` must be positive but need not be an integer. These are realisations of the algebraic identities

$$(x+1)^{n+1} = x(x+1)^n + (x+1)^n \quad \text{and} \quad \Sigma(x+1)^{n+1} = (\Sigma(x+1)^n) + x(x+1)^n + (x+1)^n$$

Next, two small variations in `Triangle` allow the Stirling numbers of the first and second kinds to be generated by replacing the verb `n` with `s1` and `s2` as indicated below.

```
s1=.#@]        NB. Replace n with s1 to obtain Stirling nos. of first kind
s2=:>:@i.@s1   NB. For S. nos. of 2nd kind, also add parens .. ((s2 * Row), a ..
```

Stirling numbers of the second kind give the number of partitions of n objects into r non-empty subsets, and those of the first kind give the numbers of partitions into non-empty cycles. More details can be found in Howard's reference [0]. The usage is `(1 1,:0 0)&Triangle Repeat m`. Since the verb `n` is being replaced, the first of the four integers in the left argument is a dummy, the others must be 1, 0 and 0 as shown.

Using VideoSoft FlexGrid Pro with DyalogAPL: an Update

by Jon Sandles

Introduction

In the last issue of Vector I wrote an article describing how to use the VideoSoft FlexGrid with Dyalog APL. I reported on a number of shortcomings, by far the worst of which was Dyalog's incomplete handling of events coming from the ActiveX control. Before Vector had even landed on your doormats, Dyadic had published a patch for their new version 8.2 interpreter.

As well as fixing this bug, the new 8.2 interpreter has also completely cleared up the instability problems I was having previously.

The fix is available for download (if you are a member of Dyadic's DSS service).

QueueEvents

In my last article I showed how you should be able to write the following event handler in the 'BeforeEdit' event of the flexgrid.

```
      ∇ msg←onFGBeforeEdit msg
[1]     :If 'Sales'≡(1⊃msg)⎕WG'TextMatrix[0;',(⍕msg[4]),']'
          ⋄ msg[5]←1 ⋄ :EndIf
      ∇
```

The setting of `msg[5]` to 1 should set the 'Cancel' parameter to 'True'. This should reject the event. Because results were not returned to the ActiveX control under Dyalog APL this technique did not work.

Dyadic offered the following explanation on their WWW pages:

> "By default, events generated by an ActiveX control (associated with an instance of an OCXClass object) are processed like all other events in Dyalog APL.
>
> This means that if you attach a callback function to an event in an OCXClass object, the events are queued up when they are received and then processed one-by-one, by `⎕DQ`, from the internal queue. This is the mechanism used to process all events in Dyalog APL and it has many advantages:

> Events are handled in an orderly manner
>
> An event cannot interrupt a callback that is processing a previous event
>
> Incoming events are held up so that you can trace a callback function etc.
>
> The disadvantage of this approach is that, for internal reasons, your APL callback function is unable to return a result to the ActiveX control, or to modify any of the arguments supplied by the event. This is a severe problem if the ActiveX control relies on callbacks to control certain aspects of its functionality.
>
> The new QueueEvents property allows you to change the normal behaviour so that it is possible for a callback function to return a result to an ActiveX control."

So by simply adding the following line to my *Go* function:

```
fg ⎕WS'QueueEvents' 0
```

And Bingo! Suddenly the edit events are being rejected. Great stuff. The other handler that didn't work - the one that was meant to set the little scroll tips on when you move the scrollbar up and down -

```
      ∇ msg←onFGBeforeScrollTip msg
[1]     (1⊃msg)⎕WS'ScrollTipText'('Row : ',▼msg[3])
      ∇
```

- also suddenly started working. So Dyadic have solved the problem completely.

Conclusion

Dyadic have acted promptly to address the issues raised in my last Vector article and as a result the FlexGrid is a viable alternative grid control for use under Dyalog APL.

Memory Mapped Files in J

by Chris Burke

J 4.02 supports memory-mapped files. A memory-mapped file can be associated with a J noun — referencing the noun is equivalent to reading the file; changing the noun's value changes the file.

Some benefits:

- There is no need for explicit file i/o operations.
- Memory-mapped files can be much larger than system RAM, yet they can be accessed without page thrashing.
- Accessing memory-mapped files is very efficient, and faster than a normal file read.
- A file can be memory-mapped by two processes on the same machine, allowing them to share data.

For example, a demo of memory-mapped files we have given at conferences and presentations uses a CD-ROM that the New York Stock Exchange distributes each month with detailed trade and quote information. J nouns are mapped to 7 files with a combined size of 650MB, and these nouns can be used just like any other J noun. Talk about big arrays!

Memory-mapped files are an essential part of Windows 95/98 and NT (Win32), where they are used to implement virtual memory management. Thus, the new facility in J just hooks into code that is already part of the operating system.

This note will give an overview of memory-mapped files, and how they are defined in J. The basic ideas are pretty simple, though undoubtedly the underlying Win32 code is complex.

Win32 Memory Management

To understand memory-mapped files, we first need a basic understanding of how memory is organized in Win32. In particular, when a process such as J references a particular memory location, how is that reference translated into a reference to physical memory?

It turns out that the translation is done in two steps — the memory reference is first translated into a file reference (i.e. a file storage location on the hard disk), and from then into a physical memory reference. Let's look at the steps in turn.

Note: in the following discussion, the term "memory reference", while singular, will typically refer to a block of memory. For example, if you read a single byte from a file, Win32 in fact will read in the block containing this byte. Typically, the minimum size block Win32 works with is a page, which is a block of 4KB.

Process Memory

When a process such as J is started, it is allocated a private *virtual address space* (VAS) of 4GB (2^32). All memory references for a process are made within its VAS. The VAS is the working memory for the process, as well as the only way by which the process can communicate with the outside world.

In theory, a process can use any 4-byte integer as a memory pointer, though in practice, Win32 imposes some restrictions on what may be used. For example, Win NT disallows write access to memory used by system DLLs; Win 95/98 does not, and hence is less stable.

The VAS is *private* in that a process may not access another process's VAS; moreover, if process A references memory location X, and process B also references memory location X, typically the values will be different.

The VAS is *virtual* in that the memory locations do not correspond directly with physical storage. A simple consequence is that the process itself knows nothing about where the physical data resides on the machine, and hence has no direct control over physical memory. As a user you may be aware that your machine has so many bytes RAM (and cache), and hard drive — but the process knows only about the VAS it has been allocated.

Initially, the VAS is empty — if the process tries to access an arbitrary memory address this will likely result in an access violation. A process can only use memory addresses that are in use and therefore valid, and Win32 keeps track of these memory addresses. The process itself can allocate memory in the VAS, and Win32 also allocates some addresses on behalf the process.

For example, Win32 allocates memory addresses for the executables (J.EXE) and any related files (J.DLL), plus the Windows system DLLs. Note that the VAS needs to have these addresses in order that the process can uses the files.

File References

Now we get to the really interesting part — Win32 associates *all* valid memory addresses in a VAS with file storage. Vice versa, when a process needs to access a file, then, if not already done so, Win32 first allocates a memory address in the VAS for that file.

In the case of files such as J.EXE, there is an obvious association — the memory address Win32 allocates for J.EXE is associated with the J.EXE file itself, and the same is true for the Windows system DLLs. But what about when a process itself allocates memory?

It turns out that Win32 maintains a file on the hard disk called the *paging file*. When a process allocates memory, a equal amount of space is allocated in the paging file, and the memory-address in the VAS is associated with the address of the corresponding space in the paging file.

Therefore, we now have the first part of the memory address translation — when a process references a valid memory address, Win32 simply looks up the corresponding file address.

Physical Memory

Physical memory comes in various flavours: on-chip cache, off-chip cache, and system memory. As far as we are concerned, all this memory is treated the same. Also, just as for a VAS, Win32 associates each page of physical memory in use with a file reference.

This gives us the second part of the memory address translation — after the file reference is determined, Win32 simply looks to see if there is a corresponding physical memory address. If so, it is used; if not (i.e. the file reference is not currently associated with physical memory), Win32 creates the association. If necessary, Win32 makes RAM available for the new association by deleting existing associations (and first updating the file system if the RAM contents have changed).

In summary:

- Each process has its own virtual memory address space (VAS).
- All used memory in the VAS is associated with file storage.
- All used physical memory is also associated with file storage.

- When a process references memory in its VAS, Win32 translates this to a file reference, then translates this to a physical memory reference, creating the association if required.

Memory-Mapped Files

The mapping between VAS memory addresses and file references is known as *memory mapping*.

A memory-mapped file is simply a file for which there is an associated memory reference in some process's VAS. Thus after J is loaded, the files J.EXE and J.DLL, as well as the Windows system DLLs are all memory-mapped files.

Now suppose you read a file in J, for example:

```
dat=: fread 'profile.ijs'
```

In order for the file to be read, Win32 must first associate the file with a memory address in the VAS. The "file read" then becomes a memory copy, starting from that memory address.

The result of the read is assigned to a variable `dat`. This variable in turn must be stored in the VAS, and hence must have a file reference associated with it (in the page file).

Thus, in this example, two memory-mappings have been used — one for the file `profile.ijs`, and one for the variable `dat` in the paging file.

Note that the contents of `profile.ijs` were physically copied from the hard disk to RAM. Also, if at some later stage Win32 needs to page out this part of RAM, then the contents will again be copied, this time to the paging file.

Memory Mapping in J

Now we come to the crux of the matter.

We can simplify this process — instead of doing the file read, we can have Win32 simply memory-map the file, i.e. associate the file with a memory address in the VAS, and we can then create a J name `dat` which points to that memory address. As far as J is concerned, this would have essentially the same affect as before — the variable dat would point to a file location representing the contents of the file. In the first case, the file reference for this data is in the Win32 paging file and the actual data in RAM or in the paging file; in the second case, the file reference is the actual file reference for `profile.ijs` and the data remains on file.

This second method explicitly creates a memory-mapped file in J.

Clearly, the second method is more efficient than the first, since no actual data is copied.

Housekeeping

Quite a bit of housekeeping is needed to make all this work.

First, the Win32 API provides functions for explicitly mapping a file to a memory address in the VAS. These are not discussed here — instead see the lab *Mapped Files*.

But we also need some way of creating a J noun (here `dat`), whose data address is the memory address of the file in the VAS. To allow this, changes were made for J402.

J Noun

A J noun has the following parts:

- a *header*, that gives the type, rank, shape, address of the data, and other information
- the *data* itself
- a *locale entry* (symbol table entry) to address the header.

In versions of J prior to J402, the header and data were always stored in contiguous memory, but in J402, the header and data can be stored in separate locations. Also, J402 has utilities that allow you to create a noun's header and locale entry. This now gives us the ability to associate a name with a memory-mapped file:

- First use the Win32 API facilities to memory-map a file, returning the file's VAS memory address.
- Then use utilities provided in J402 to create a noun's header and corresponding locale entry. The address of the data in the header is the VAS address for the memory-mapped file.

After this is done, the noun appears just like any other noun in J.

When you create the new noun, you also specify the datatype. A character datatype will work for any file, but you can also specify any other datatype if the

file is in the right format, for example, a file of integer data should have 4 bytes per element, with the value for each n being `|. (4#256) #: n`. Similarly, you can specify the shape of the data.

The above describes a file which only contains data, but you can also memory-map a file which contains both a header and data. In this case the file is self-describing and you need only specify the filename.

For detailed information on the J implementation, and an example of a large memory-mapped database , see the labs *Mapped Files*, and *Mapped Files Database*.

Drag and Drop in Dyalog APL

by Jon Sandles

Introduction

When you design modern Windows applications, there is often a user-driven requirement to be able to perform complex tasks using the mouse. (As experienced programmers we tend to be keyboard freaks and frequently ignore these rodent fans. Less experienced programmers tend to have the reverse problem!) More often than not the user of the mouse is required to move a control by holding the left mouse button down and moving it - "dragging" - and then to release the mouse button over some other area of the screen - "dropping". The other area of the screen is not limited to the current form, or even the current application - it could be anywhere.

This madness was almost certainly brought to us by the people at Apple - and for once I am not thanking them. Users demand to drag and drop anything onto anything and get some sort of sensible result. For Microsoft the Windows Explorer paradigm kicked it all off and ever since then we have been dragging and dropping like mad.

This article explores how drag and drop is implemented within the Windows programming environment and shows how to mimic this behaviour in Dyalog APL. The techniques use low level Windows Operating System calls so should apply to most programming languages.

OLE Drag and Drop

The dragging and dropping of data between applications has been made possible by OLE. (Microsoft keeps changing what OLE means so I will not even start to try and explain: you just have to believe.) OLE defines a method of storing some data in global memory and registering what the data represents so that applications can quickly work out whether they understand what it is or not. There are lots of standard OLE data types that are very useful - text data, delimited text data, file types, images etc. These are very similar to the kind of global descriptors that the Windows clipboard understands. (Indeed, the keyboard shortcut to a drag and drop operation is frequently cut and paste.) If your application understands these data types then you can easily build in drop zones to your forms so that the user

can drag this data from other applications onto yours. This saves having to write complex import and export routines.

This technique is fraught with problems. Each application may have a slightly different understanding of how to treat each standard data type and hence data can become distorted or lost when transferred in this way. (Ever tried copying a bitmap from Word to Excel to Paintbrush to CorelDraw?)

As nice as it is, I will not be going any further into the issues of defining OLE drag and drop data types and storing global data. Instead, I will concentrate on the mechanisms of drag and drop **within** Dyalog APL. Because we are concentrating on this single environment I can get away with using global APL variables to share data for the drag and drop operation. This means that we cannot drag the data onto another application, nor can we accept dropped data from another application. The techniques I propose will be extensible to do this, and it would be interesting to extend this article at some point in the future.

Dragging

The mechanics of dragging and dropping are the same whatever the control, whatever the programming language. They have to be if they are to co-operate with each other.

So, let's start at the beginning. What happens to make a drag operation begin. Try it in Windows Explorer and see for yourself. The ListView control in the right hand pane of Explorer is made up of items, which are either folders or files. You can place the mouse over one of these ListView items and press the left mouse button down and without letting the button back up you can start to drag the object around the screen. If the mouse button comes back up, the drag operation ends. Thus, we need to catch the mouse button down event and the mouse move event and check that the mouse button stays down. As soon as we detect the mouse button has come up we must abort the operation.

This doesn't turn out to be as simple as it sounds. Firstly, you need to take care that you start the drag correctly. Try the following code...

```
     ∇ TestDrag
[1]    Init  ⍝ just a placeholder to contain ⎕na defintions _ see later
[2]    'f'⎕WC'form'
[3]    'f.tree1'⎕WC'treeview'('event'('MouseMove' 'onMM'))
[4]    'f.tree2'⎕WC'treeview'('event'('MouseMove' 'onMM'))
[5]    'f.s'⎕WC'Splitter' 'f.tree1' 'f.tree2'
[6]    'f.tree1'⎕WS'items'(,⊂'a' 'b' 'c')
[7]    'f.tree2'⎕WS'items'(,⊂'1' '2' '3')
[8]    ⍙handles←('f.tree1' 'f.tree2')⎕WG⊂'handle'
[9]    ⍙controls←'f.tree1' 'f.tree2'
     ∇


     ∇ msg←onMM msg;down
[1]    down←1=5⊃msg
[2]    :If down
[3]        msg
[4]    :EndIf
     ∇
```

Run this code and you get a form with two panes (like Explorer). Both are empty at the moment. Try doing a drag in the left hand pane (on a mythical pixel!) and watch the event messages that appear in the session as you drag the mouse around. Notice how all the mouse events go to the control where the drag started (the source control) and the position of the mouse is given relative to that source control.

Clearly, we have quite a lot of work to do to be able to work out where the mouse is and which control it is dragging over or dropping on (the target control). To do this we have to take control of the drag from the source and direct the mouse move messages to the relevant target controls. At the same time we can adjust the cursor to indicate the state that the drag is in. (Have another look at Explorer and see how many different cursor states you can get by dragging over different things.)

Mouse Capture

The first step is to capture the mouse. This technique ensures that all mouse messages are sent to the source control and all positions are relative to the source control (the Dyalog mouse move event is sort of doing this already, but unfortunately not for all objects, so to be safe we take control of this ourselves). You want to apply this as soon as the drag has begun, which is the first mouse move following the mouse down, and remove the capture on the mouse up. We need two ⎕NA calls:

```
⎕NA'I user32.C32|SetCapture I'
```

```
⎕NA'I user32.C32|ReleaseCapture'
```

We can then adjust the onMM event to use the SetCapture call. (Note how the event handlers are removed for the duration of the drag - we need to use our own and then we need to put the old handlers back when the drag has terminated.) When the drag is in progress the cursor should change to a no-entry symbol. (A sensible default as the source does not know yet whether there are any valid targets to drop on. At the start of the drag we need to identify the data that is to be dragged. In this example, I will take it as the currently selected items in the tree that the mouse button was over. If there are none selected we should not even enter the drag operation. (I can't think of any reason why dragging nothing around is functionally useful!) At the end of the drag (when the mouse goes up) we need a new handler (attached at the start of the drag - onDragUp) to end the drag process (doing whatever we decide it should do) and by calling ReleaseCapture.

```
     ∇ msg←onMM msg;down;sink;selected
[1]    down←1=5⊃msg
[2]    :If down
[3]        ∆source←1⊃msg
[4]        selected←∆source ⎕WG'selitems'
[5]        :If ∨/selected
[6]            ∆data←⊃selected/∆source ⎕WG'items'
[7]            ∆events←∆source ⎕WG'event'
[8]            sink←SetCapture ∆source ⎕WG'handle'
[9]            ∆source ⎕WS'event'('MouseMove' 'onDragMove')('MouseUp' 'onDragUp'
[10]           ∆cursor←'.'⎕WG'cursor'
[11]           '.'⎕WS'cursor' 12 ∧ No entry cursor
[12]       :EndIf
[13]   :EndIf
     ∇

     ∇ onDragMove
[1]    'DragMove'
     ∇

     ∇ onDragUp;sink
[1]    'DragUp'
[2]    sink←ReleaseCapture
[3]    ∆source ⎕WS'event'('MouseMove' 0)('MouseUp' 0)
[4]    ∆source ⎕WS'event'∆events
[5]    ∆data←''
[6]    '.'⎕WS'cursor'∆cursor
     ∇
```

If you try this code you will find that dragging off the empty tree view results in a "no-entry" cursor and a stream of 'DragMove' messages followed by a single

'DragUp' message. Do not worry – this is all it's meant to do. The next step is to extend onDragMove to give more feedback about the drag process. We need to identify which control the mouse is over and what the position of the mouse is relative to that control. If the control that the mouse is over is a valid drop target we will ask it whether we can drop the source data onto the control at the mouse's position. We need three more ⎕NA calls to achieve this:

```
⎕NA'I user32.C32|WindowFromPoint I I'

⎕NA'I user32.C32|GetCursorPos >I[2]'

⎕NA'I user32.C32|ScreenToClient I =I[2]'
```

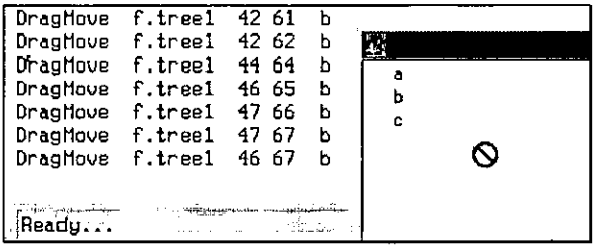
We use GetCursorPos to retrieve the position of the mouse (safer to get it from the Operating System) and then WindowFromPoint to find out which window is underneath that point. This is returned as a window handle, and we then convert the absolute point co-ordinate we just got to be relative to the window handle we just got by using ScreenToClient.

Because this code returns the control we are over as a window handle rather than a Dyalog control name, we need to store the control names and window handles of any valid drop targets on creation. Add the following two lines to the main function DragTest:

```
[8]     ∆handles←('f.tree1' 'f.tree2')⎕WG"='handle'
[9]     ∆controls←'f.tree1' 'f.tree2'

      ∇ onDragMove;posn;key;handle;target
[1]     posn←⊃⍵GetCursorPos 2
[2]     handle←WindowFromPoint posn
[3]     posn←⍵⊃⍵ScreenToClient handle posn
[4]     :If handle∈∆handles
[5]         target←(∆handles⍳handle)⊃∆controls
[6]         'DragMove'target posn
[7]     :EndIf
      ∇
```

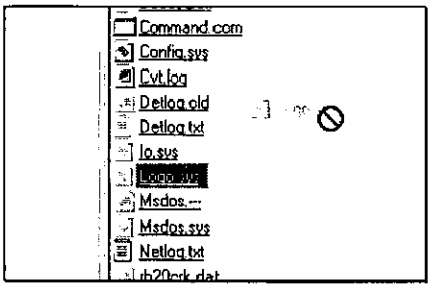
DragMove checks that the window handle we are over is one of the drop targets we have created and if it is it displays the true control name and relative position on the screen. Phew! This may seem like a lot of work to do nothing, but we are pretty well getting there. The basic principle of capturing the mouse and discovering which window the drag is over and then reporting the drop to that window has been put in place. Try it:



To go any further we need to think about what we are dragging, where we are dragging it and what we want to do when it gets dropped.

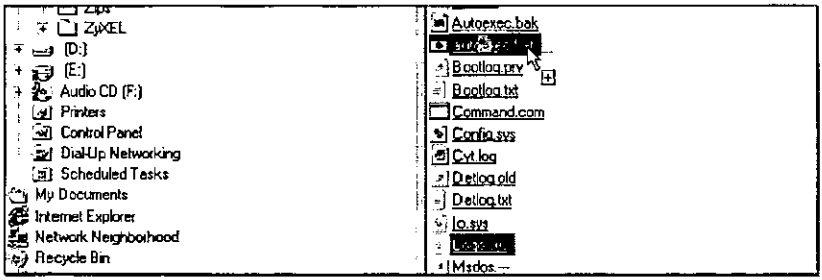
Hit Tests and Dropping

Have another look at Windows Explorer. Try dragging a file a short distance across the list view in the right hand pane.



What happens? You see a no entry cursor! Why is that? Well, its because you are not allowed to drop the file in the same directory that it came from, that would be silly. Do the same again, but this time when you are half way through the drag depress the <ctrl> or <shift> key whilst still dragging. The no entry cursor changes to a drag-copy cursor (modified with the "+"). This is because you are allowed to drop a copy of the file into the same directory and <ctrl> or <shift> is the Windows standard for modifying the drag from a move to a copy. It is important to support this functionality, if the copy operation makes as much

sense in the particular context. Sometimes only the copy operation makes sense, in which case you don't need to code this behaviour: the normal drag operation performs a copy and the <ctrl>/<shift> key has no effect. Indeed, if the drag operation you are performing is a copy operation then it is semantically best to always use the drag copy cursor, or modify your own cursor with the "+". (You can also combine <ctrl>/<shift> to make it a drag-link operation - but let's not overcomplicate the example.) Now try dragging a file over another file that is associated with an executable extension (.exe, .bat or .com).



Notice how the drag magically modifies to a drag-copy. If you drop, the file will be passed as a parameter to the executable. (Precisely how is defined in the Windows registry, potentially for each executable.) I view the use of the drag-copy cursor in this situation as being a bit overloaded, but it is better than no indication at all. The point is that the type of data underneath the mouse is allowed to modify the drag operation in order to guide the user.

How is all this behaviour handled internally? Our DragMove function needs to perform these tricks somehow. We already know what control is underneath us at any time, but it would be impossible to write generic code for every control and circumstance under the sun, so we must ask the control to perform this functionality itself. So that we know what is going on we ask all controls, who want to partake, to register themselves as drop targets on initialisation. If they aren't registered we won't pass any drag events to them. In this case we will just keep displaying the no entry cursor to indicate that a drop would be rejected. Line [7] of DragTest already registers the controls required, I now add a line which registers the drag functions that will be called whenever a drag event occurs over that object. In this test case, both controls will support the same behaviour so I use the same function twice (as a general rule you need a function per control).

```
[10]    ⍙dragfns←'#.Drag' '#.Drag'
```

This function gets passed the following arguments by the DragMove function.

TargObj	The object being dragged over
SourceObj	The object that owns the drag data
Posn	the position within *TargObj* that we are currently dragging
DragData	the data being dragged
DropFlag	1 = it was actually dropped

This function (#.Drag) is responsible for doing any drop hi-liting on the target object (drop hi-liting is the process by which you indicate which part of the object the mouse is currently over – it is not the same as selecting the item – indeed the selection is left intact when drop hi-liting is done). The process of drop hi-liting is very specific to the particular windows control you are implementing it for. In our example , with the TreeView, the drop hi-lited item is indicated in the same way as a selected item (a little confusing, but that's Microsoft for you). An item is drop hi-lited whenever the mouse is over it. We detect which item in the tree we are over by running a hit-test. If we are over an item we drop hi-lite it. If we are no longer over any items or if we are over a different control we must clear all the drop hi-liting (otherwise you cannot tell which item is selected anymore). To do all this we need two more Windows API calls:

```
'smht'⎕NA'I4 user32.C32|SendMessageA I4 U I4 =(I[2] U I4)'

'smsh'⎕NA'I4 user32.C32|SendMessageA I4 U I4 U'
```

'smht' is a cover to allow us to send the Windows API message TVM_HITTEST to the tree view. To cut a long story short the message returns which tree view item the mouse is currently over and can be wrapped up as follows:

```
      ∇ r←form HitTest posn;whdl
[1]     whdl←form ⎕WG'handle'
[2]     r←smht whdl(4352+17)0(posn 0 0)
[3]     r←1⊃r
      ∇
```

'smsh' is a cover to allow us to send the Windows API message TVM_SELECTITEM (Dyadic provide a cover for this but it doesn't seem to work correctly). You pass this event along with the value TVGN_DROPHILITE and the particular item to hi-lite. If the item is 0 then all hi-liting is cleared.

This is wrapped as follows:

```
      ∇ form SetItemHiLite item;whdl;sink
[1]     whdl←form ⎕WG'handle'
[2]     sink←smsh whdl(4352+11)8 item
      ∇
```

Because these API calls deal in terms of Tree View "items" we need to be able to convert these into something that DyalogAPL will understand. Fortunately, there is another Windows API message to convert the item into the index of the item in the tree. This is:

```
'smgih'⎕NA'I4 user32.C32|SendMessageA I4 U I4 ={U U U U I4 I4
I4 I4 I4 I4}'
```

This allows us to send the Windows message TVM_GETITEM which allows us to get the full details of the item in a TVITEM structure. As it happens the tenth item of this structure is a user-defined data attribute, and it turns out that Dyalog APL stores the index of the item in the tree in this item. So we take advantage of this (admittedly undocumented and hence dodgy) fact and use the following wrapper:

```
      ∇ r←form FindItemHandle hdl;whdl
[1]     whdl←form ⎕WG'handle'
[2]     r←smgih whdl(4352+12)0(4,hdl,8ρ0)
[3]     r←1+10⊃2⊃r
      ∇
```

This is all we need to proceed with the hit-test process of the drag. We can now implement our #.Drag function to do the hit-test and hi-lite the item that is hit:

```
      ∇ r←targobj Drag arg;sourceobj;posn;objdata;dropflag;item;ix
[1]     r←0
[2]     sourceobj objdata posn dropflag←arg
[3]     :If sourceobj≠targobj
[4]         sourceobj SetItemHiLite 0
[5]     :Else
[6]         (⍎controls~targobj)SetItemHiLite¨0
[7]     :EndIf
[8]     :If 0<item←targobj HitTest posn
[9]         targobj SetItemHiLite item
[10]        ix←targobj FindItemHandle item
[11]        r←1
[12]    :Else
[13]        targobj SetItemHiLite 0
[14]        r←0
[15]    :EndIf
```

```
[16]
[17]    :If (r>0)∧(dropflag)
[18]        'data'objdata'from'sourceobj'dropped on'ix'in'targobj
[19]    :EndIf
      ∇
```

Do not worry too much about all the SetItemHiLite 0 lines - they just make sure the drop hi-liting is cleared whenever the mouse has gone out of scope. This function returns a value of 1 if the drag copy is over a valid drop target and a valued of 0 if it is not. It could potentially have returned a 2 to indicate that the drag move has been modified to be a drag copy operation. Our DragMove handler can be modified to call this function as follows:

```
      ∇ onDragMove;posn;key;handle;target;fn;mode
[1]     posn←⊃⍎GetCursorPos 2
[2]     handle←WindowFromPoint posn
[3]     posn←⍎⊃⍎ScreenToClient handle posn
[4]     :If handle∈Δhandles
[5]         target←(Δhandles⍳handle)⊃Δcontrols
[6]         fn←(Δhandles⍳handle)⊃Δdragfns
[7]         mode←target(⍎fn)Δsource Δdata posn 0
[8]         :Select mode
[9]         :Case 1
[10]            '.'⎕WS'cursor' 'drag'
[11]        :Case 2
[12]            '.'⎕WS'cursor' 'dragplus'
[13]        :Else
[14]            '.'⎕WS'cursor' 12
[15]        :EndSelect
[16]    :EndIf
      ∇
```

'drag' and 'dragplus' are the standard Windows cursors for a drag and a drag copy operation, which should be defined in the Init function we wrote earlier. Of course, it is at this point that you realise you need the same piece of code when the drop occurs in onDragUp. The only difference being the 4th argument to our drag function should be set to 1. Hence create a wrapper for this piece of code, effectively by renaming onDragMove to:

```
      ∇ DragDrop drop;posn;key;handle;target;fn;mode
```

And then changing line [7] to:

```
[7]     mode←target(⍎fn)Δsource Δdata posn drop
```

OnDragMove becomes:

```
      ∇ onDragMove
[1]     DragDrop 0
      ∇
```

And onDragUp becomes:

```
      ∇ onDragUp;sink
[1]     DragDrop 1
[2]     ⍙controls SetItemHiLite⍠0
[3]     sink←ReleaseCapture
[4]     ⍙source ⎕WS'event'('MouseMove' 0)('MouseUp' 0)
[5]     ⍙source ⎕WS'event'⍙events
[6]     ⍙data←''
[7]     '.'⎕WS'cursor'⍙cursor
      ∇
```

That is all there is to it! You should now have everything you need to build quite complex drag and drop operations. We have concentrated on trees as they are one of the most common controls to code drag and drop and also one of the most interesting. The architecture that has been used is easily extensible to plug in new controls without effecting the overall structure of the code. If you have made it this far, I suggest you experiment further with Windows Explorer to see the kind of hit-testing is done, and then try and implement your own variations.

It will be around now that you will start to realise there is (at least) one major component missing from what I have demonstrated – keyboard handling.

Keyboard Extensions and Auto-scrolling

This essentially mouse-driven operation is also very dependent on the keyboard to provide certain modifications to the process. The most important key by far is the <Esc> key - which provides a way of aborting the drag operation without having to let go of the mouse and risk the operation actually doing something unintended!

As mentioned earlier, the <Ctrl> and <Shift> keys can also be used to modify the state of the drag operation. But these keys by themselves do not generate keypress events in Dyalog APL so we have to be a lot cleverer. You can use a Windows API call to get the current keyboard state:

```
⎕NA'I2 user32.C32|GetKeyState I'
```

The escape key is defined as VK_ESCAPE (or 27). This function returns a negative value if the key is down and 0 if it's up. Also VK_SHIFT and VK_CONTROL can be queried. (16 and 17). Alternatively, you can use the GetKeyboardState API call to list the state of all 256 keys simultaneously.

```
⎕NA'I2 user32.C32|GetKeyboardState >I1[256]'
```

We could use these API calls in our onDragMove function to detect the keyboard state during each mouse move, but this is not good enough: an <Esc> or a <Ctrl> modifier can occur when the mouse is not moving. Try Windows Explorer again to see how it works. You should be able to keep the mouse perfectly still during a drag and use <Ctrl> to toggle the copy/move state. Given these keys do not trigger keypress events - how do we do this? We have to use a timer. The timer must be fast enough that the effect is good enough, but not unnecessarily fast otherwise processor cycles will just be wasted (every 50 milliseconds is ample). This new code is included as follows. Add the timer on starting the drag:

```
[10] 'f.timer'⎕WC'timer'('Interval' 50)('Active' 1)('Event' 'Timer' 'onTimer')
```

Then code the timer to query the keyboard and if anything has changed to call onDragMove or if <Esc> is down to abort the drag.

```
      ∇ onTimer msg;oldstate
[1]     :If 0>GetKeyState 27
[2]         EndDrag
[3]     :Else
[4]         oldstate←keystate
[5]         keystate←GetKeyState 17
[6]         :If oldstate≠keystate
[7]             onDragMove
[8]         :EndIf
[9]     :EndIf
      ∇
```

Modify Drag to use the global keystate to upgrade the drag state from move to copy.

```
[11] r←1+keystate<0
```

This is enough of an example to give you an idea. All the standard keys need to be handled as well, often in combination with each other. The complexity can become quite mind-boggling.

Summary

Drag and drop is standard Windows functionality. If the development environment you are working in does not support drag and drop natively then there are Windows API calls that allow you to achieve the required effects. These are fairly easy to use in isolation but when used in combination with each other the task soon becomes reasonably complex.

A solution is to provide a generic abstraction layer that encapsulates the common functionality required for all controls and queries each control for the control specific information. Once this layer is in place the task is considerably simplified.

Some of the more common API calls have been investigated in order to establish the general principles involved. There are so many different types of control and so many different application specific requirements that the architecture has been deliberately left open. Tools such as Visual Basic wrap the common layer in a simple object model and most controls comply to this simple API. This is what is required to simplify the task within APL.

References

Duncan Pearson: Various conversations and workspaces have changed hands over the years!

"OLE Drag and Drop" - MSDN Visual Studio 6 help

At Play with J: New Model Computer

by Gene McDonnell

Donald Knuth began writing his series called *The Art of Computer Programming* in 1962, and devised a mythical computer called MIX with which to illustrate computer language programs, and as a machine to be used for exercises. Half a lifetime has gone by since then — half a human lifetime, three or four computer lifetimes. His MIX design is now obsolete. A new machine, called MMIX, will replace it in subsequent revisions of his volumes. A preliminary writeup of this new machine is given in Knuth's web site.

His home page address is http://sunburn.stanford.edu/~knuth.

Briefly, MMIX operates on 64-bit words. It has 256 general-purpose 64-bit registers that can hold either fixed point or floating point numbers, or characters, or arbitrary binary data. Most instructions have the form OP X Y Z, where OP, X, Y, and Z are each 8-bit bytes. If OP is the ADD instruction, for example, the meaning is "X=Y+Z", that is, "Set register X to the contents of register Y plus the contents of register Z." There are 256 op codes that fall into a dozen or so categories. The virtual memory available to an application is 2^64 bytes. There are no input-output instructions; files are handled as memory-mapped data. There is no operating system at the moment for MMIX. Knuth writes, "Whenever I have been asked if I will be writing a book about operating systems, my reply has always been 'Nix.' Therefore the name of MMIX's operating system, NNIX, should come as no surprise."

The purpose of this column is to discuss a few J-related aspects of this machine. I am attempting just now to write a J model of it. Time will tell if and how well I do this.

On this machine numbers are 64 bits long, though the design tolerates shorter numbers. For floating point numbers it uses the IEEE/ANSI standard 64-bit form, and accommodates the 32-bit forms only to transform them to the 64-bit form in computations. More interestingly, integer numbers are also 64 bits long. Thus an unsigned integer can take on any value from zero through 18,446,744,073,709,551,615, or approximately 1.8e19. Signed integers range between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807. One of the reasons I chose J as simulation language is its extended integers, which allow me to use these large numbers directly, with the least amount of fuss: an occasional terminal 'x' on a number, or 'x:' preceding a number.

Many years ago I heard Luther Woodrum, the one who gave APL a good grade (he designed and implemented the grade primitives in APL\360) ask, "When will computer designers put in a maximum instruction?" In almost all computers, finding the maximum of two numbers involves a branch instruction. Something like this:

```
condition is x is less than y

branch to Label if condition is true

maximum is x

branch to exit

Label: maximum is y

exit:
```

One of the great motivations of computer designers is to reduce the number of branches required by programs executing at the machine level. Branches really slow things down. I recently attended, along with Larry Breed (the key designer and implemetor of APL\360), a talk Knuth gave on MMIX. I had read the preliminary MMIX writeup and hadn't found a maximum instruction, so after the lecture, in the question period, I asked how one found a maximum on MMIX. Knuth fumbled a bit, and gave an explanation which I didn't hear very well. Larry, however seemed to understand and accept it, so I didn't pursue the issue then, but the next day I phoned Larry and asked him for more. He explained that there was an identity that could be used to obtain maximum at the machine level without a branch, namely:

```
a max b  is  b + 0 max a - b
```

I said to Larry that that looked a bit circular, didn't it, defining maximum in terms of itself? Larry assured me that the circularity is only apparent. He said also that, before retiring from IBM a few years ago, he had been instrumental in getting the architects of an IBM computer (I think the RS6000) to add a "difference or zero" instruction, primarily to handle maximum, and that now Knuth had devised a similar feature for MMIX. It's called 'zero or set if negative', written ZSNI rX, rY, Z, which acts as follows: If register Y is negative, register X is set to Z, otherwise nothing happens. To see how this works in finding maximum, look at the three cases below. The two values are in A and B and the result is to appear in B.

```
                  case 1        case 2        case 3
                 A=8   B=3     A=3   B=8     A=3   B=3
SUB   A,A,B       5     3      -5     8       0     3   A=A-B
ZSNI  A,A,0       5     3       0     8       0     3   A=0 max A
ADD   B,A,B       5     8       0     8       0     3   B=A+B
```

Somewhere below the machine instruction level there is a branch, but the machine is happy nevertheless, because the three instructions above execute very rapidly, and in a pipelined machine the pipeline is undisturbed, since no instruction-level branch is needed.

The next thing I'd like to mention is exemplified by a frustrating aspect of IEEE floating point as implemented by two different systems: the little-endian and the big-endian school. In the little-endian implementations the bytes are numbered from left to right, 7 through 0, and the bytes in the big-endian school are numbered 0 through 7. This leads to strange things like, in J, looking at the value of a floating point number on a Macintosh and on an Intel machine gives two different pictures, one reversed from the other. Another bit of explanation I need to give you is that Knuth uses the prefix # in front of a number expressed in hexadecimal. One of the 'bit fiddling' instructions is 'multiple or', denoted by MOR. The explanation given for it is:

Suppose the 64 bits of registers Y and Z are indexed by pairs such that the bytes are numbered from 0 to 7 and the bits within the bytes also by 0 through 7; then the MOR operation replaces bit ij of register X by the bit

$$y_{0j}z_{i0} \text{ or } y_{1j}z_{i1} \text{ or } \ldots \text{ or } y_{7j}z_{7i}$$

Thus, for example, if register Z contains #0102040810204080, MOR reverses the order of the bytes in register Y, converting between little-endian and big-endian addressing.

What Knuth has designed, if the above is opaque to you, is a special case of the good old-fashioned 'or dot and' inner product. When I came to describing this instruction in my simulation, in addition to standard housekeeping, all I had to write was

```
U =. B +./ . *. A
```

where B and A are 8-by-8 boolean matrices representing the 64 bits in the two operands.

If I get anywhere with this simulation, I'll probably report on it in a future column.

Utilities for Dyalog APL/W Developers: "UCMD" Take 2

by Ray Cannon

(All the code relating to this article is available from the Vector web site http://www.vector.org.uk as a zipped file of about 0.6Mb).

I have recently reviewed how my original "pseudo UCMD" code (Ref.: *Vector* 15.1 and 15.3) worked, and have now simplified it resulting in version 2.

Recap of Version 1

The utility suite consisted of a directory (called UCMD) containing many workspaces, one for each utility and one extra workspace (called "SE.DWS") which is run each time Dyalog is loaded, to set up the environment for the current session.

Each utility workspace contained at least one and sometimes two (or more) top-level functions with the same name as the workspace name. (*e.g.* workspace "FNS.DWS" might contain two top-level functions "Fns" and "FnS".) Niladic utilities appeared on the menu bar, while monadic and dyadic utilities appeared in a drop down list from the tool bar.

Version 2

Modifications made to the environment

Enhanced function key support, including multiple alternative key settings, and PFKey editing

Improved menu bar layout and structure

Improved access to HELP.

Modifications required for "Dyalog 8.2"

Under Dyalog 8.2, the simple Dyalog menu-bar and tool-bar have been replaced by cool-bars. "UCMD" Version 2 has been modified to include support for this new environment. (Thanks to Bob Hoekstra for all code changes relating to Dyalog 8.2.)

Modifications made to the "UCMD" process.

Under version 1, all the control of which "UCMD" utilities were available, was hard coded within the session set-up workspace "SE.DWS". So every time a utility was added, removed or renamed, "SE.DWS" also had to be modified.

Under version 2, "SE.DWS" has no in-built knowledge of what utilities are available. Instead, it checks each workspace in the "UCMD" directory for the existence of a standard function ("*UCMDSETUP*" see below) which, if found, it uses to build a data-base of the utility workspaces currently available.

For convenience, utilities are now (loosely) grouped together by "CLASS".

Monadic and Dyadic utilities are still available from the tool-bar drop down combo, but niladic utilities are now accessed (by default) via a single drop down menu called "UCMD". In addition, each "UCMD" workspace has its own help.

Some of the new utilities that have been added

- "Explore+" - a "pan workspace" namespace explorer. Edit, copy and compare functions, across workspaces and namespaces
- "Store" - stores the current object back into a workspace. Like the ")STORE wsid fn" on old STSC mainframe APL.
- "Memorise" - copies the contents of temporary storage (see old utilities "Put", "ListGot" and "GetAll") onto file.
- "Recall" - copies the Memorised data back into the active workspace.
- "Forget" - erases the Memorised file.

  ("Put" and "GetAll" only worked within a single Dyalog session. "Memorise" and "Recall" can be used between sessions.)
- "PutTree" - puts a copy of the current object and all functions in its calling tree into temporary storage (see old utilities "Put", "ListGot" and "GetAll").
- "EditPfk" - edits the current PFKeys settings
- "ChangePf" - switch between different sets of PFKeys settings.
- "Setpf" - sets up the PFKeys.
- "CompareDws" - compares two Dyalog workspaces held in DWS files. (Similar to "WsCompare", which compared the currently loaded WS with a single DWS file.)
- "EditSe" - A "UCMD" set-up utility, allowing you to re-configure several aspects of the "SE"/"UCMD" environment.

How to Install Version 2 of "UCMD"

If you already have version 1 installed, please remove or rename the "UCMD" directory before installing this new version. This new code assumes that you use a standard Dyalog session (such as "def_uk.se") which creates both a menu bar and a toolbar. It has not been tested on non-standard or altered sessions, and may require modifications before it will work with them.

You can download the file "UCMD.ZIP" from http://www.vector.org.uk

In your Dyalog WS directory create a new directory "UCMD" (for example "C:\Dyalog73\ws\UCMD") and unzip the contents of "UCMD.ZIP".

Start up Dyalog and (optionally) add the new "UCMD" directory in the Dyalog workspace search path.

To make these utilities available your session just load the "SE.DWS" workspace. I always auto-load the SE workspace, and never save the modified session.

The "SE.DWS" workspace looks for the command line parameter "loadws=<wsid>", and if found will attempt to auto-load the named workspace after it has finished initialising itself.

The "*UCMDSETUP*" function.

If you are considering adding your own favourite utility into the "UCMD" environment, you will need to save it as a workspace in the UCMD directory. In addition, the workspace must contain a "*UCMDSETUP*" function used by "SE.DWS" to set up the menu/tool bars. Look at a few examples from directory UCMD. (The following is taken from "FNS.DWS"):

```
    ∇ {r}←UCMDSETUP menu;helpfid;helpno;name;text;fn1;hint;fn2;left;right;
      class;readme
[1]   ⍝Add command to menu and combo
[2]   name←'fns' ⍝Unique menu item name. (Menu order sorted by this value)
[3]   class←'LIST'
[4]
[5]
[6]   fn1←'FnS'      ⍝ Must be a function in the workspace or '' if N/A
[7]   text←'&Fns'    ⍝ Text that appears in the menu       or '' if N/A
[8]   hint←'List fns in current namespace.' ⍝ Hint text    or '' if N/A
[9]
[10]  fn2←'Fns'                  ⍝ Must be a function in the workspace or ''
[11]  right←'Criteria eg foo*'   ⍝ Description of Right hand argument
[12]  left←'Optional Namespace'  ⍝ Descr of Left hand argument or '' if MONADIC
[13]
```

```
[14]   helpfid←'ucmd.hlp' ⍝ Help file name or '' if None Available
[15]   helpno←'700'       ⍝ Help context number or '' if None Available
[16]   readme←c'FNS.DWS --  Lists some/all of the Fs in the namespace (plus)'
[17]   readme,←c''
[18]   readme,←c'The menu command FnS lists all the functions (and operators)'
[19]   readme,←c'in the current namespace.'
[20]   readme,←c''
[21]   readme,←c'The combo command Fns lists only thoses function that match '
[22]   readme,←c'the entered text (with "*" acting as a "wild card").'
[23]   readme,←c''
[24]   readme,←c'If an optional namespace is supplied, the resulting listing'
[25]   readme,←c'include the full namespace path (for ease of editing).'
[26]   readme←↑readme
[27]   :If 0≠⎕NC'menu'
[28]       r←menu RunSetup name class fn1 text hint fn2 right left helpfid helpno
[29]   :Else
[30]       r←hint readme class
[31]   :EndIf
       ∇
```

Notes

- "class" groups related utilities. *e.g.* "Put" and "GetAll", "Fns" and "Vars";
- "name" must be unique.
- "fn1" if supplied, MUST be the name of a niladic function in the workspace with the same name (ignoring case). It should be empty if you do not want the item to appear on the "UCMD" menu;
- "text" is the text that will appear in the menu. It may contain spaces. Indicate the accelerator key (if required) via an ampersand. May only be empty if "fn1" is also empty;
- "fn2", "right" and "left" relate to monadic and dyadic utilities;
- "fn2" may be empty if you do not want the item to appear in the drop down combo. If supplied it MUST be a monadic (or greater) function in the workspace with the same name (ignoring case);
- "right" contains the text that will be displayed when assigning the right hand argument. May only be empty if "fn2" is also empty;
- "left" must either contain the text that will be displayed when assigning the left-hand argument or must be empty if "fn2" is monadic or empty.
- "readme" defines the text used to create simple "HELP" messages.

FileCheck

Another Utility for the Dyalog Session

by Bob Hoekstra (Bob.Hoekstra@khamsin.demon.co.uk)

Introduction

Following on from Ray Cannon's recent articles [1, 2], I thought readers might like to know about one of the tools I have created to assist my development work in Dyalog APL.

I'm sure that I can't be the only APLer who regularly gets irritated by the hassle of looking into component files. ... *Was it in the third component? ... Do I have to write a vector or a 1-row matrix? ... etc.*

To reduce this source of irritation, I have found `UCMD\FILECHK.DWS` to be increasingly useful, hopefully others will also! My requirements were as follows:

1. A component file would be share tied with as little fuss as possible, and be untied when I'm done.
2. It must work from the session as well as from the file manager.
3. It must have no residual effect on the session/workspace.
4. It must display the file name, file size and number of components
5. It must allow easy opening and closing of other component files.
6. It must display the contents of each component in such a way as to make the structure clear.
7. It must allow "alternative views" (to be explained later) of certain components.
8. It must never confuse me (tall order).
9. It must be reasonably fast and easy to use.

In Use From Your APL Session

All my productivity tools now work as part of Ray's `UCMD` suite although they may also be used as stand-alone utilities.

I have `FileCheck` set to appear on my menu bar, and it also appears in the UCMD menu structure. Figure 0 shows this.

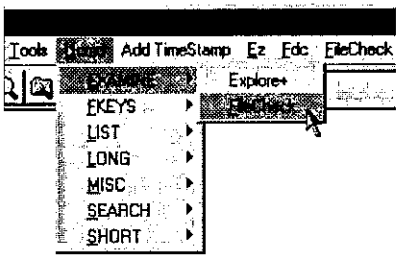


Fig 0: `FileCheck` in the APL session UCMD menu structure

Selecting the utility will import the code into □*SE* and run it, much as described below. There are just a few special conditions to running from the session:

1. The `Component File Inspector` window will open immediately (see figure 1).

2. If you have no files tied, you are immediately prompted with the standard file open dialogue to select one.

3. If you have files (share or exclusively) tied, you will not be prompted for another file. You may however elect to open another file using the appropriate button or menu. Note that the file name combo box will appear to be empty – pull it down and select the file you would like to examine before doing anything else.

4. On exiting, any files you had open will not be closed, but files opened by FileCheck will be. Of course, if you forced a file to close (by pressing the Close File button – see figure 1) this file will not be re-opened for you.

5. The system is modal. The session is "dead" until FileCheck has been closed.

In Use From Your File Manager

I have created a file, `FCDEMO.DCF`, to demonstrate the working utility. Double clicking on this file in the file manager of your choice immediately starts the `Component File Inspector` (CFI) window (see fig. 1).

In my case, I'm using Windows NT Explorer, but this can also be made to work from the Windows 3.x File Manager or the Windows 95/98 equivalent, as well as *dtfile* and *filemgr* (if you're using CDE or OpenLook on a supported UNIX operating system).

Files are share tied by the run-time interpreter. To set this facility up so it works from Windows NT Explorer, start Explorer, select View/Options followed by the File Types tab. There should be an "APL Component File" registered file type (extension DCF) to edit, but if not, create it. Create (or edit) the action "open", which must include an "application" line like:

```
<DEVICE>:\<PATH>\dyalogrt.exe <DEVICE>:\<PATH>\filechk OPENFILE="%1"
```

Users of other operating systems can figure out from the above line how to create their command string.

The Component File Inspector

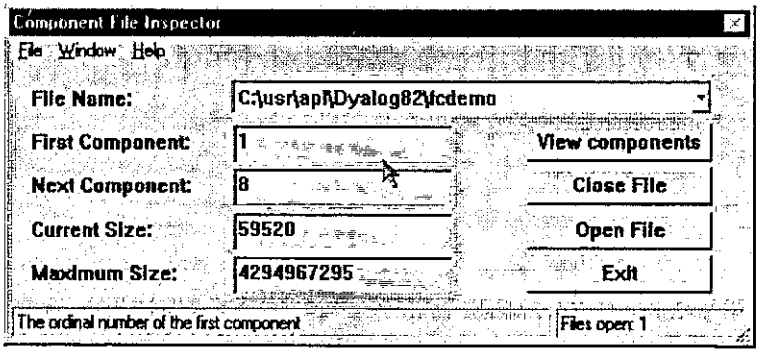


Fig 1: Component File Inspector window

The CFI has:

- A menu bar for those that like it. All functions (except the Help/About) are available from the buttons on the CFI.
- A file name drop-down combo object. This allows you to select another open file (if you have more than one open). The bottom option on the combo is always blank, and selecting this has the same effect as pressing the Open File button (or selecting File/Open from the menu bar).
- Four display only edit objects which show the ordinal number of the first component and that of the next component to be appended to the file, and the current and maximum sizes of the file.
- Buttons which have the following actions:
  - Viewing the components (also available as Window/View on the menu bar).

- Closing (i.e. untying) the currently selected file (also available as File/Close on the menu bar).
- Opening (i.e. tying) another file (also available as File/Open on the menu bar).
- Exiting the application (also available as File/Exit on the menu bar).

- A status bar, with a status field for hints and another to let you know how many files you have open.

Only two of the above controls need any further explanation. Firstly, opening another file brings up the standard file open dialogue from which you merely choose the file you wish to open.

Secondly, and the main thrust of the whole exercise, is the viewing of a component:

The Component Viewer

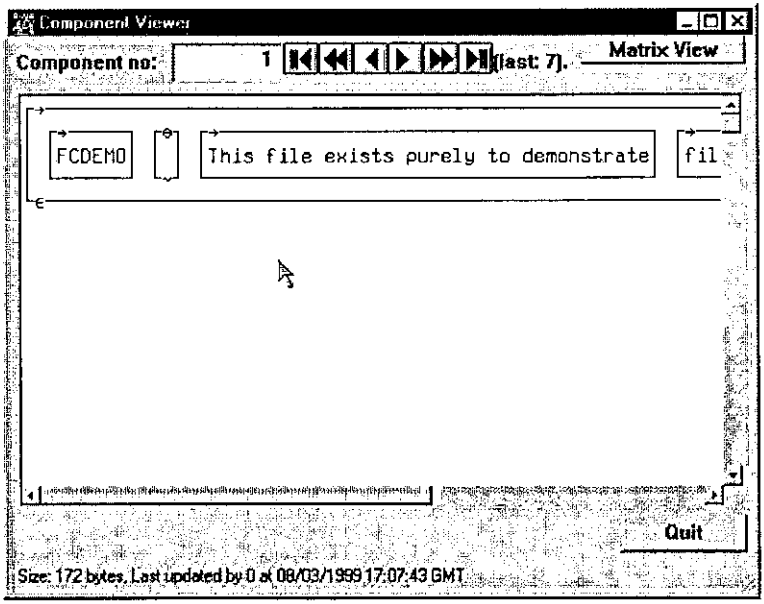


Fig 2: Component Viewer

Figure 2 shows the first component of the demonstration file, a vector of character vectors. Note the status field, which tells us the size and the time of last update of

this component. For those viewing this in monochrome, the text is blue on a white background. Note also the prescence of the "Matrix View" button, which I press to produce figure 3 below.

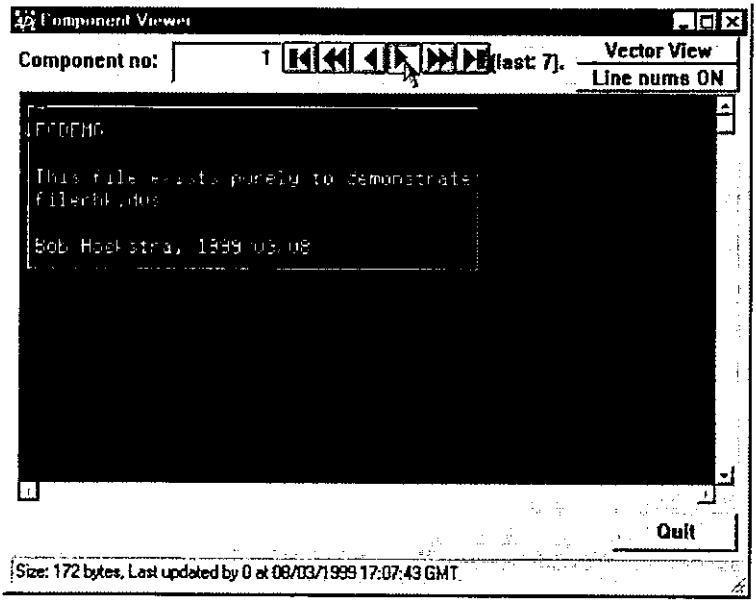


Fig 3: Component Viewer showing Matrix View

In figure 3 the mouse is depressing the "video control" to go to the next component and the text is white on a blue background to indicate that the contents of this component is not displayed in its "natural state". Whenever a matrix is being displayed, line numbering may be switched on. The "Vector View" button would revert to the "natural state display", but so would choosing another component. So, without further ado, let us now progress to the second component, the "index table" for the file, which I show in figure 4 with line numbering switched on and the window resized vertically to show the whole index:

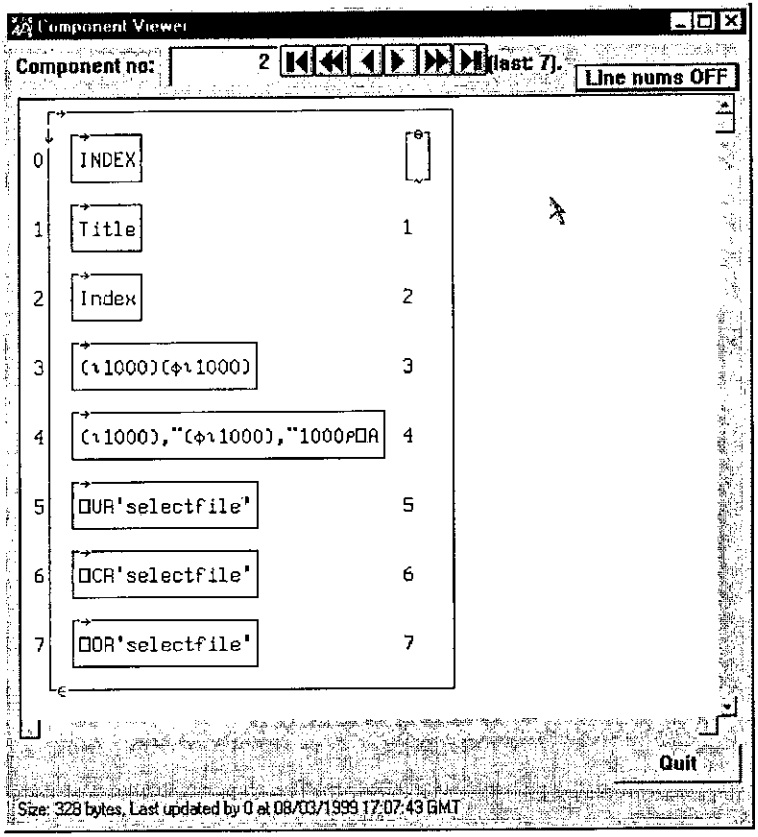


Fig 4: Component Viewer showing Line Numbering

Moving on to component 3 (figure 5, resized vertically again), we see a new control on the window. The display has been broken up into "chunks". The reason for this is that Microsoft apparently will not let us display more than 1024 characters without imposing its own form of wrapping, which does nothing for the clarity of the result. Hence "chunks", the first few of which are the maximum 1024 characters wide, with the last one taking up the slack.

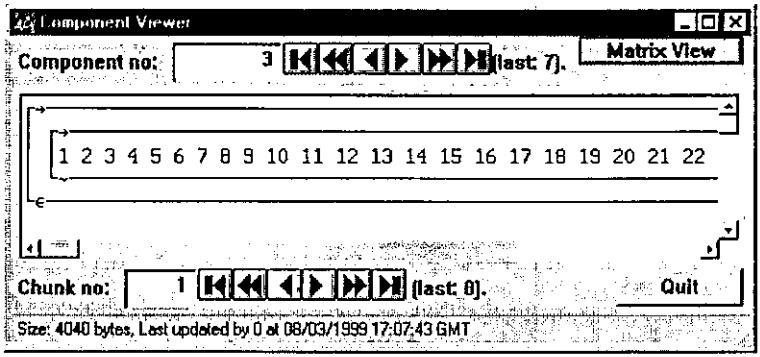


Fig 5: Component Viewer showing Chunks

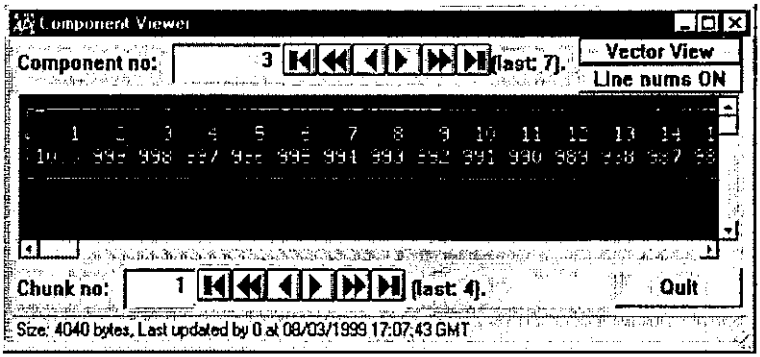


Fig 6: Component Viewer showing reduced Chunking by a Matrix View

Referring to the index, we can see immediately that this is a case for the "matrix view", hence figure 6 above. Not only does this clearly indicate the positional relationship between the two vectors, but it reduces the "chunking effect" as we would have expected. This effect is even more marked with component 4, 12 chunks initially, but no chunking using the matrix view.

Of course, this does not justify the existence of the "matrix view". I believe that it can, in many cases, produce a clearer view of the data in the file. This is shown in figures 5 and 6 above, as well as in figures 7 and 8 below.

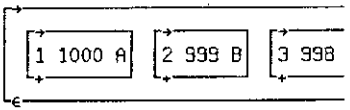


Fig 7: Natural view of Component 4 (12 chunks)

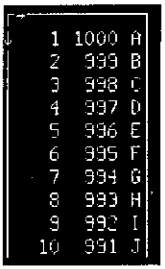


Fig 8: Matrix view of Component 4 (no chunks)

Lastly, there are three representations of the same function in the FCDEMO file. The *Vector Representation* in component 5 is clearly a vector in figure 9, while the Canonical Representation in component 6 is clearly a matrix in figure 10.

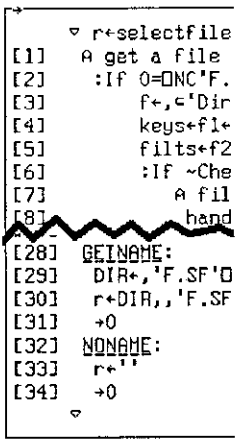


Fig 9: Vector Representation of a function

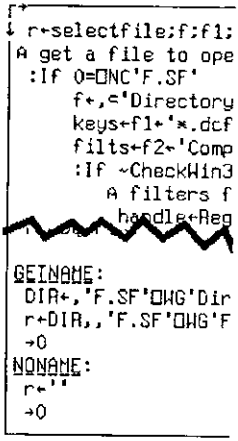


Fig 10: Canonical Representation of a function

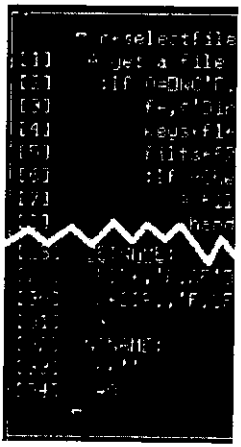


Fig 11: Object Representation of a function

The *Object Representation* (component 7, figure 11) was originally a bit of a problem, as the resulting special object (not your normal APL variable) could not be treated by the display function (I use the dynamic version, not the normal DISPLAY function, as it provides, to me at least, more pleasing output and is a little faster). The code reacts to such a failure by formatting the object and running the result through display again and changing the display colour to red on black. In this case (a function) the result looks like a Vector Representation, but when

viewing a Object Representation of a name space or GUI object, only the original (fully qualified) name is displayed.

Video Controls

The video buttons (which I have almost ignored) work as follows for both components and chunks:

- The single arrows move to the previous (pointing left) or next (pointing right) item. Having reached the first/last item, they will wrap on a further push and go to the last/first item.
- The extreme left and right buttons (arrow pointing towards a double bar) go to the first or last item, with no "wrap".
- The "double arrow" buttons move in steps calculated to traverse the entire range in about five clicks.

File Filters

The file open dialogue uses filters of "Component Files(*.dcf)" (the default) and "All Files(*.*)". Users of Windows NT can extend this by including filters in the registry. My registry entry is described below:

```
[HKEY_CURRENT_USER\Software\Dyadic\FileCheck\Filters]
"KEYS"="*.dcf *.obj *.pch *.pln *.*"
"*.dcf"="Component Files(*.dcf)"
"*.obj"="APL Object Files(*.obj)"
"*.pch"="APL patch files(*.pch)"
"*.*"="All Files(*.*)"
"*.pln"="AYP plan files(*.pln)"
```

How to Get FileCheck

FileCheck is included in the zip file on the *Vector* web site containing Ray's UCMD workspaces. The address is http://www.vector.org.uk/v151/quadse.zip. Alternatively the zipped workspace can be downloaded from my web site at http://www.khamsin.demon.co.uk/sware/filechk/filechk.zip.

References

1. Utilities for Dyalog Developers, by Ray Cannon, *Vector* Vol. 15 No. 1
2. Utilities for the Dyalog Session (part 2), by Ray Cannon, *Vector* Vol. 15 No. 3

Functional Extensions to the Fibonacci Sequence

by John Sullivan

Joseph De Kerf returns to the subject of Fibonacci's sequence in Vol.15 No.3[1]. In his article he gives a function for the calculation of the terms of the Fibonacci sequence which only uses the first term of the golden section formula and rounds off:

```
      ∇ F←FIB1 N;R
[1]     F←⌊0.5+(÷R)×(0.5×1+R←5×0.5)*N
      ∇
```

The Fibonacci sequence, like others defined by recurrence relations, is normally defined for positive integers only

$$F_n = \begin{cases} 1 & \text{if } n = 1 \text{ or } 2 \\ F_{n-2} + F_{n-1} & \text{if } n > 2 \end{cases}$$

or for non-negative integers

$$F_n = \begin{cases} n & \text{if } n = 0 \text{ or } 1 \\ F_{n-2} + F_{n-1} & \text{if } n > 1 \end{cases}$$

depending on whose textbook you read, but I prefer to define it as the second way with "if $n > 1$" replaced by "otherwise", which, of course, allows negative values. The values for negative n are the same in absolute value as those for positive n, but the signs alternate. In order to generate values for negative n as well as positive n you will need both terms of the golden section formula:

```
      ∇ F←FIB2 N;R
[1]     F←⌊0.5+(÷R)×((0.5×1+R)*N)-(0.5×1-R←5*0.5)*N
      ∇
```

We can see the difference between these two functions as follows:

```
        FIB1 ¯4 ¯3 ¯2 ¯1 0 1 2 3 4
0 0 0 0 0 1 1 2 3
        FIB2 ¯4 ¯3 ¯2 ¯1 0 1 2 3 4
¯3 2 ¯1 1 0 1 1 2 3
```

What Now?

Having extended the "normal" domain of the Fibonacci sequence to the negative integers, the most natural follow-up question is "Can we extend the Fibonacci sequence to the real or complex numbers?" and, unsurprisingly, the answer is "yes". What we are after is a function $f(x)$ with $x \in \mathbb{R}$ or $x \in \mathbb{C}$, such that $f(x)$ takes the Fibonacci values when x is a real integer, and $f(x) = f(x-2) + f(x-1)$ for all x. There are one or two mathematical considerations to be overcome in converting the golden section formula into a function of a continuous variable, but the process is straightforward. The main points are as follows.

A real function

Before we can turn integer n into real x, we need to get rid of the negative term in the nth term formula. We can do the following:

$$\frac{1-\sqrt{5}}{2} = \frac{(1-\sqrt{5})(1+\sqrt{5})}{2(1+\sqrt{5})} = \frac{-4}{2(1+\sqrt{5})} = -\frac{2}{(1+\sqrt{5})}$$

so, if we let $a = \frac{1}{2}(1+\sqrt{5})$ we obtain $F_n = \frac{1}{\sqrt{5}}(a^n - (-a)^{-n})$.

Since n is an integer, $(-a)^{-n} = (-1)^n a^{-n}$, and we note that $(-1)^n$ has the same values as $\cos n\pi$ for all n. Now we can replace integer n with real x, define $a^x = e^{x \log a}$, and obtain a continuous real function: $f(x) = \frac{1}{\sqrt{5}}(a^x - a^{-x}\cos \pi x)$.

The real-valued Fibonacci function in APL is very easy:

```
      ∇ z←Fibfn x;s
[1]     z←0.5×1+s←5*0.5
[2]     z←((z*x)-(2○○x)×z*-x)÷s
      ∇
```

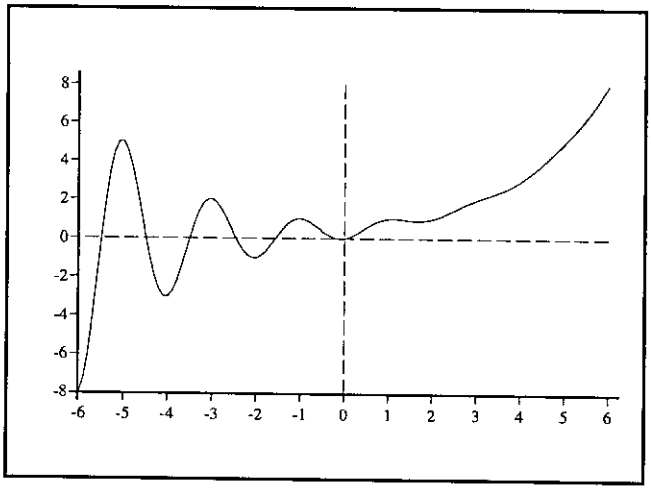
The complex case

The complex case is a bit easier, because we can use the golden section formula direct. In order to write an APL function for the complex case, we need a suite of appropriate complex arithmetic functions. I decided to amend a suite published by Prof. A Hawkes [2], rather than write my own. The complex Fibonacci function looks like this:

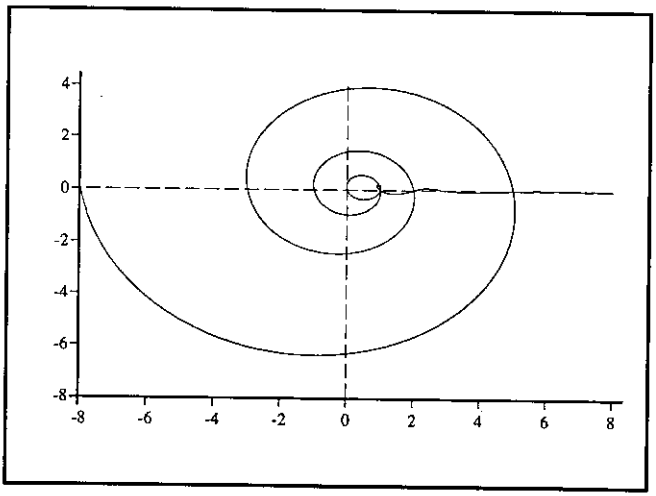
```
      ∇ z←FibfnC x;s;ns
[1]     x←COMPLEX x
[2]     z←COMPLEX 0.5×1+s←5*0.5
[3]     z←((z EXP x)-(-z)EXP-x)DIV COMPLEX s
      ∇
```

What do they look like?

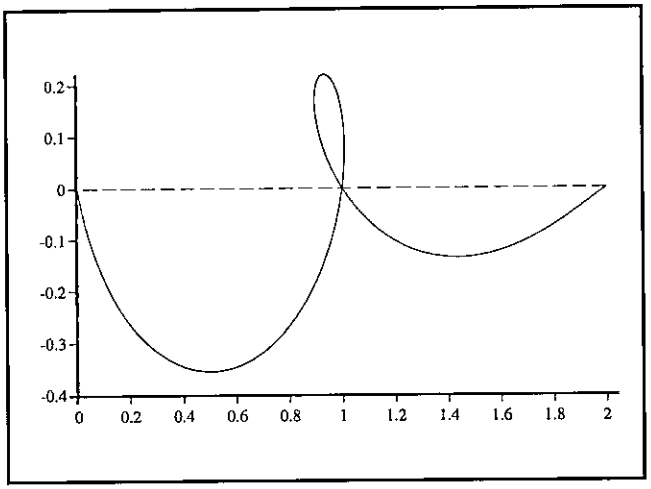
Here is a graph of the real function. Note that the curve to the left of the origin is a quite interesting variant on the damped sine wave, as the following extract showing $f(x)$ against x between $x = \pm 6$ shows.



For the complex function we can plot the imaginary part against the real part for real x between ± 6:



As it is a bit difficult to see what is going on when $x = 1$, here is an enlargement of the part where $0 \leq x \leq 3$



References and Acknowledgements

[1] J. De Kerf, *From Fibonacci to Horadam*, *Vector*, Vol.15 No.3, p.122.

[2] Alan G. Hawkes, *Complex Numbers in APL*, *Vector*, Vol.1, No.1, pp.107–120.

All the graphs were drawn with *Rain* publication graphics for APL, available from Causeway Graphical Systems Ltd.

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, YO62 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 YO62 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.

1998/99 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	Stefano Lanzavecchia stf@adaytum.dk	c/o Vector Administration, Brook House, Gilling East YORK YO62 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@csi.com	138 Burton Stone Lane, YORK YO30 6DF
Education	Dr Ian Clark 01388-605544 100021.3073@compuserve.com	Unit R21, Auckland New Business Centre, St. Helen Auckland, Bishop Auckland, Co. Durham DL14 9TX
Administration	Rowena Small 0181-980 7870 treas.apl@bcs.org.uk	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Stefano Lanzavecchia	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 Hockstra (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 7DP
Tel: 01653-696760
Fax: 01653-697719
Email: causeway@csi.com
Web: www.causeway.co.uk

Compass Ltd
Compass House
60 Priestley Road
GUILDFORD, Surrey GU2 5YU
Tel: 01483-514500

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants, RG24 0AL
Tel: 01256-811125
Fax: 01256-811130
Email: sales@dyadic.com
Web: www.dyadic.com

HMW Trading Systems Ltd
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel: 0171-353 8900
Fax: 0171-353 3325
Email: 100020.2632@compuserve.com

Insight Systems ApS
Nordre Strandvej 119C
DK-3150 Hellebæk
Denmark
Tel: +45 49 76 20 20
Fax: +45 49 76 20 30
Email: info@insight.dk
Web: www.insight.dk

APL2000
Rapid Application Development
6610 Rockledge Drive
Suite 502
Bethesda MD 20817
USA
Email: sales@apl2000.com
Web: www.apl2000.com

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