VECTOR

APL News, Reviews and Tips ...

BCS

The Journal of the British APL Association

ISSN 0955-1433
www.vector.org.uk
Vol.19 No.1 July 2002

A Specialist Group of the British Computer Society

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 or via email. APL code can be accepted 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 (any version).

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 2002–2003

Category	Fee	Vectors	Passes
UK Private	£20	1	1
Overseas Private	£22	1	1
(Supplement for Airmail, not needed for Europe)	£4		
UK Corporate Membership	£100	5	5
Overseas Corporate	£135	5	
Sustaining	£500	10	5
Non-voting Member (Student, OAP, unemployed)	£10	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, Vector Production, Brook House, Gilling East, YORK YO62 4JJ. Tel: 01439-788385.

Email: apl385@compuserve.com.

Contents

dyalog APL

for Microsoft .net

Thanks to Dyadic's ground-floor involvement in Microsoft's .NET project, Dyalog.Net is fully integrated with .NET and gives you access to all its features, just like other .NET languages. Better still, **its APL!**

This is your chance to beat VB programmers at their own game!

- Fully integrated with .NET class libraries
- Write new .NET class libraries in APL
- Publish APL web pages, integrated with IIS
- Implement XML Web Services in APL
- Call XML Web Services from APL
- Includes APLScript scripting language
- Save workspaces as DLLs and .EXEs
- and lots more ...

Dyalog.Net

http://www.dyadic.com

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

by Stefano Lanzavecchia

"If Lisp is not a mainstream language, why are we using it as the framework for our discussion of programming? Because the language possesses unique features that make it an excellent medium for studying important programming constructs and data structures and for relating them to the linguistic features that support them. [...] Above and beyond these considerations, programming in Lisp is great fun."

Structure and Interpretation of Computer Programs – A. J. Perlis

Some people love to collect stamps, some others prefer empty bottles of beer, some have a soft spot for dolls and some would be dead without their weekly intake of comics. I adore programming languages. Thanks to the internet I can indulge in my passion of finding new programming languages, studying them, possibly even try them and learn as much as I can not so much about the language itself, but about the reasons that justified its creation, its peculiarities, its unique strengths. Most important of all, in fact, it's the attempt at learning new techniques to solving problems, techniques that may prove useful in other contexts. This hobby of mine is not particularly new. It dates back the days when I first discovered that computers had programming languages that I could learn. It exploded like a disease when in my university days, I first used a terminal with a connection to the Internet. I was really amazed when I found out that there were so many programming languages that one page couldn't list them all and, more, that new ones were being added all the time and that (oh joy!) of many of them existed completely free implementations. At the time I was about to get my master in Physics so I should have taken a more practical approach to computing, but because of the morbid fascination I had, I just fell in love with any new language, useful or useless. Now that I am a professional developer I have somehow adjusted my aim and my sensitivity: what I look for when I come across a new language is the thrill of the unexpected and the untried but also some sort of applicability. The meaning of "new" is of course "yet unknown to me", since in the last 3 years, for the first time I've come across languages already some 25 years old, and still very interesting. It's a somewhat late realisation the meaning I give to "applicable".

When I tried to collect ideas for this editorial, I discovered much to my surprise, that what I am looking for in a language to make me consider it interesting for anything more than academic theories (which I also enjoy a lot) is an extensive set of utility functions to perform what are normally considered trivial tasks or, otherwise, to communicate with the outside world. An example should make this foggy definitions clearer. We have a collections of files and we want, without the aid of archiving utilities, spread them over a set of floppy disks, in such a way that the slack empty space left is minimal. If you think that the problem is somewhat outdated, then let's substitute the words "floppy disk" with "CD-ROM" to have a problem I have to solve at least once a week. Let's imagine that we found an amazing programming languages whose engine is based on a really cool optimiser, with a simple and intuitive syntax which makes writing the solution almost trivial. Alas, this dream language doesn't have in its standard library a function to read directories from storage facilities. True, it can read ASCII files, so we could redirect the output of a *dir* command to a file and post-process it, but it starts being messy and our (well, my) excitement for the language has decreased and our trust has subconsciously started to fade.

Here's a suggestion for your next rainy Sunday afternoon. I suspect that if you live in England you won't have to wait long for it to arrive even if it's summer... Download the documentation for the standard library set distributed with languages like Perl, Python, Ruby, Java, SmallTalk, Visual Basic, C#, SML, a good implementation of Forth, Delphi or Erlang. Not only you'll find that they have functions or objects or modules, depending on the paradigm used, to give you complete control over the files on your Hard-Disks, starting of course from the list and their sizes, but that they have XML parsers, SOAP producers and consumers, ODBC bindings, matchers for regular expressions, lexers, parsers, network bindings, advanced GUI building capabilities, CGI extensions. They can interface to mail servers as well as web servers. They can talk to your DNS, they let you interface to 3D graphical libraries, they can play music, they let you write with minimal (OK, this last is a lie, but only just) effort distributed applications. And if the mentioned capabilities are not in the standard library set, then there's at least one website which collects public extensions to the library where it doesn't take long to find a working implementation of the exotic protocol that you have to interface to. For some languages it's possible that the implementation of the funny protocol is a commercial one, but it's definitely there and not even that expensive.

What about APL?

I could stop here, but it wouldn't be fair. I am not the first one to lament a serious lack of a library of APL workspaces to accomplish the most trivial tasks as well as the sophisticated ones. Not surprisingly, around the most recently born APL dialects a library is being built, thanks to the support of the vendor itself, but also thanks to many volunteers. It's almost heart-warming to see a complete regular expression engine in the standard J library, as well as fairly complete OpenGL and ODBC bindings. It's nice to see a powerful set of functions to produce graphics. It's cool to have basic bindings to network services. There's still no trace of SMTP, POP3, FTP, HTTP, LDAP (to mention a few popular internet protocols) support, no XML, no SOAP, no ORBA, no compression, no encryption, but it's clear that things are moving in the right direction for the J community. I am not blaming Dyadic or Cognos because they don't invest enough in libraries: they are market-driven and they do what they can with their resources.

The blame goes to all those who complain about the poor acceptance level of APL and yet stick to the tradition of leaving things as they are. I am amongst those. In a recent issue of Vector, Adrian and I showed how it was possible to use a free library to do data compression. We could have spent time refining the solution, and packaging it as a ready-to-use set of functions but using as excuse the chronic lack of time of a professional, we left it at the stage of proof of concept. Nobody took it from there to complete it, though, so we are in good company.

Actually, the kind of refinement I am talking about is not something one person can do on his own. First, there need to be versions for at least all the Windows' interpreters available. Second, the proposed solution must be used by other parties who must send feedback to the original author, possibly even send improved code and documentation. And yet it all must remain freely available for public consumption. Impossible? That's how the impressive libraries accompanying Perl , Python, Ruby (and so on) grew. People had a problem to solve, did it themselves, did not accept the idea of their work being duplicated over and over, wanted to show-off and donated it. Over the years a brook carves a canyon... And the project in which the original library was used, improved in quality because of the works of the others who took over the component. There's always somebody more talented than you are in at least one area and it's a waste not to let him help you if he's willing to.

We all have our small set of functions to read a directory list. Some of us built interesting operators which are the equivalent of an each on a list of files, recursively generated to span an entire subtree. Why don't we refine them and submit them to a public forum? I know the answer already: no time to refine anything "I only do stuff I am paid for and my boss wouldn't let me donate my code", and lack of a properly moderated forum. I am not the first one to propose something like this, although I have the impression that in the past the focus was more on more basic algorithms, such as clever ways of removing trailing blanks.

That's also very interesting and it matches perfectly the spirit of the project. But it's no longer enough: to be palatable APL must be presented to the potential new user with a library to perform tasks that, even if not in the spirit of the language, are fundamental to build even the simplest complete application. Careful though: I am not saying that somebody should write an XML parser in APL. We all know that a fully compliant XML parser implemented in APL would perform so badly that nobody would want to use it. It's OK to cheat in these cases. I've already praised the beauty and the power of quadNA and foreign functions (Java, .NET). We could have a simple-minded XML parser implemented in APL to play around (there's one already implemented in K) and another one based on external bindings for mission critical applications.

Isn't it a contradiction to wanting to spread APL but not APL code? How are the future APLers supposed to learn? There are hundreds of thousands of lines of good public Perl code, code that can be used to learn Perl as well as learn about the problem whose solution is coded. Once I have understood how to extract simple information about a JPEG encoded bitmap (such as width, height and so on) by reading the source code of a JPEG decoder written in Java, actually, of the JPEG decoder provided with the standard Sun Java SDK.

I cannot do much more than to throw the idea in the arena once again hoping that the times have matured since the last proposer did the same and that the bright success stories of the other programming communities can inspire us. I can even propose myself as a coordinator *ad-interim* till a better one is found, confiding in the understanding of my boss. You know how to contact me.

Quick Reference Diary

Date	Venue	Event
22-25 July, 2002	Madrid, Spain	APL2002 Conference
November TBA	Naples, Florida	APL2000 User's Meeting

Dates for Future Issues of VECTOR

	Vol.19 No.2	Vol.19 No.3	Vol.19 No.4
Copy date	6th Sept	6th Dec	7th March
Ad booking	13th Sept	13th Dec	14th March
Ad Copy	20th Sept	20th Dec	21st March
Distribution	October 2002	January 2003	April 2003

APL2000 Inc

APL+Win 4.0 for Windows 95/98/NT/ME and 2000

- ActiveX APL Grid Object
- ? Button for "What's This?" help
- Noredraw property for visible controls
- New connection with printer dialog boxes
- Combo box supports images and indents
- Greatly increased maximum file size
- Calling syntax for ActiveX objects
- Full compliment of printed documentation

APL+LinkPro 3.0

- Link between APL+Win and non-APL databases
- Open Database Connectivity (ODBC)
- Wide variety of databases on different platforms

APL+UNIX 5.3

- Control Structures (compatible with APL+Win)
- Interface for TCP/IP communications using sockets
- Partition function (APL2 compatible)
- ⎕AT – Attributes (APL2 compatible)

APL

APL Systems IDC SL
Alfredo Marquerie, 12 – 2F
28034 Madrid, Spain
+34 91 730 7008 Voice
+1 775 743 6131 Fax
+34 60 680 5949 Cell
uksales@apl2000.net
uksupport@apl2000.net

Coming Soon! **APL+Web Services**

British APL Association News
Report and Accounts

Minutes of the meeting of the Committee of the British APL Association held on 24th May 2002 at the Royal Statistical Society.

It was agreed that the Committee would be the same as last year except that Stefano has served three years and cannot be elected (the committee co-opted him) and Stephen Taylor was co-opted to the committee to run the schools project.

The Committee re-appointed John Sullivan to audit the accounts for 2001-2002. [This didn't happen but the secretary & treasurer have deemed it did.]

Anthony Camacho
25th May 2002

Minutes of the AGM of the British APL Association held at the Royal Statistical Society from 2pm to 2.15 pm on 24th May 2002

Apologies were received from Ian Clark and Stefano Lanzavecchia. The minutes of the last AGM as printed in Vector Vol 18 No 1 were approved.

The Chairman reported as follows:

Chairman's report on the year 2000-2001

The committee for next year is:

Chairman	Adrian Smith
Secretary	Anthony Camacho
Treasurer	Nicholas Small
Editor	Stefano Lanzavecchia (co-opted)
Webmaster	Ray Cannon
Activities	Jon Sandles
Schools project	Stephen Taylor (co-opted)
Projects & publicity	Alan Mayer

The subscriptions are now £20 for an individual, £100 (with entitlement to 5 copies of Vector) for a corporate and £500 for a sustaining member.

The Association has done two things worth a mention.

The first is to keep Vector going. Vol 18 number 4 is in the post on its way to members. The production team apologises for being five weeks late and fully expects to have 19.1 on time again - it will be given to every delegate at Madrid with a special offer for two years membership. The new subscription rate means that it will come close to breaking even next year; if we could recruit some more members and advertisers it might make a profit. The imminent widespread introduction of unicode may well cause problems with the typesetting.

The second is the A+ interpreter port. The Association spent £1500 on this and made it available under Windows. It is totally primitive and lacks a decent working environment, but it works and is available for download. About 500 people so far have downloaded it. It is doubtful whether we will do further work on it as we will probably use J for the schools project.

We have offered help to APL 2002 but as it is being underwritten by the university no financial help was required. It looks as if it may be a good conference.

The Treasurer, Nicholas Small, presented the accounts (see opposite page). There were no questions.

Anthony Camacho
25 May 2002

British APL Association
Summary of Annual Accounts 2001/02

Summary of income and expenditure/receipts and payments:

	2001/02		2000/01
	(R&P)	(I&E)	(I&E)
	£	£	£
Income/Receipts			
Subscriptions	10525	9884	9736
BCS services	0	0	183
Bank interest	1703	1703	2251
Vector advertising (incl. VAT)	4431	4056	2967
Other	493	629	394
	-----	-----	-----
Total receipts	**17152**	**16272**	**15530**
	-----	-----	-----
Expenditure/Payments			
Meetings	240	240	265
Administration	1298	1348	1347
BCS services	0	0	183
Vector production and despatch	15162	15352	16344
Projects	2369	2369	1721
Other	386	308	355
	-----	-----	-----
Total payments	**19455**	**19617**	**20215**
	-----	-----	-----
Assets summary:			
Bank and other balances		37349	39652
Debtors		1121	1912
Creditors		(5516)	(5265)
		-----	-----
Net assets		**32954**	**36299**
		-----	-----

Notes.

Pence figures have been omitted, so columns may not add exactly.

The value of stocks of Vector have not been assessed, nor has the value of the Association's computing hardware and software.

For 2001/02, figures are shown both as income and expenditure, i.e. revenues strictly relating to the activities of that year, and as receipts and payments, i.e. what goes in and out of our bank account. The comparative figures for 2000/01 relate to income and expenditure.

Membership at 30.4.02 (previous year's figures in parentheses)

	UK		FOREIGN		TOTAL	
	Number	Vectors	Number	Vectors	Number	Vectors
Sustaining	6 (5)	28 (25)	3 (3)	41 (51)	9 (8)	69 (76)
Corporate*	4 (7)	29 (50)	1 (1)	10 (10)	5 (8)	39 (60)
Corp. Ind*	19 (19)	19 (19)	2 (2)	2 (2)	21 (21)	21 (21)
Individual	118 (122)	118 (122)	239 (237)	238 (235)	357 (359)	356 (357)
Non-voting	16 (16)	16 (16)	0 (0)	0 (0)	16 (16)	16 (16)
Life	1 (1)	1 (1)	1 (1)	1 (1)	2 (2)	2 (2)
Library	1 (0)	1 (0)	6 (5)	6 (5)	7 (5)	7 (5)
Russians			11 (10)	11 (10)	11 (10)	11 (10)
APL Groups			13 (13)	39 (39)	13 (13)	39 (39)
						560 (586)

*Add the Vector numbers in these rows to get the total subscribed for by corporate members

The loss of 30 individual members was almost balanced by 19 new members and 10 lapsed members who rejoined.

We have lost three corporate members, two taking 10 copies of Vector, and the Dutch APL is taking 10 fewer copies.

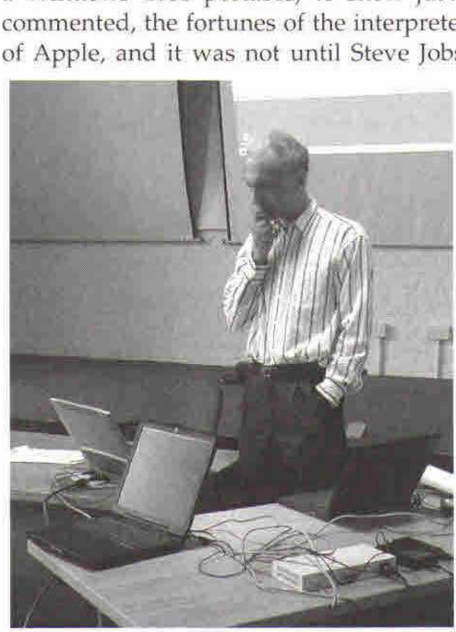
On the plus side, MicroAPL has rejoined as a sustaining member and we have two new library members.

Vendor Presentations

notes by Adrian Smith

Richard Nabavi on APL/X

APL/X is APL.68000 reborn as a cross-platform APL2-compatible interpreter. Richard used an interesting little network (with a Linux server, a PowerBook and a Windows 2000 portable) to show just how cross-platform it really is. As he commented, the fortunes of the interpreter are still very much tied to the fortunes of Apple, and it was not until Steve Jobs showed some signs of sorting out the mess at Apple that MicroAPL thought it worthwhile to invest the time in a full rebuild of the old interpreter.



He showed us around the existing interpreter, and also some of the things from the 1.1 pre-release which is due in around 6 months. This has one nice new idea (other vendors please copy) which is to be able to label control structures and then say:

```
blaaah
:leave loop3
blaah
```

To quit out from a named loop, or While block. Obviously, this works with :Continue in the same way.

John Scholes on *Mornington Crescent*

Well actually, John Scholes on pathfinding through directed graphs using Dfns. This was a fascinating half-hour, at least in part for the tutorial on how to code up the London Underground as a directed graph so you can ask the software hard questions like "How do I get from Heathrow T3 to Heathrow T4 by train?". Yes, the system has one-way sections, and not all transfers between platforms are as bi-directional as the maps would have you believe. This is crying out to be a Web-service – I hope they give it a go on Dyalog.NET soon!

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, 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
APL Borealis Inc.	Dyalog APL	poa	Distributor of Dyalog APL products from Dyadic
	APL2000	poa	Distributor of APL2000 products
APL Systems IDC SL			
	APL+Win v3.6 Full version	1895	A 32-bit Windows-hosted interpreter that runs under Windows 95/98/ME/NT/XP
	Upgrade to v3.6 from v3.x	474	
	to v3.6 from APL+Win v2.x	1034	
	to v3.6 from APL*Plus III v1.x	1249	
	Migration to v3.6 from APL*Plus II or APL+Plus PC	1464	
	APL+Unix		(Please contact us for 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.
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.
Dittrich & Partner	APL+Win	poa	Cognos/APL2000 Inc products
	Dyalog APL	poa	Dyadic Systems Ltd. products
	IBM APL products	poa	
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.
DynArray	DICE for Windows	poa	Software development kit which includes an APL interpreter as a DLL and the ability to run and link existing and new APL code to non APL code such as VB, C/C++, Java and integration with various Windows software applications and database packages such as MS Office.
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.
	Workstation APL2 V2 Version 2	$1500	AIX, Linux, Solaris, Windows Product 5724-B74, Part Number 45P7514
	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	Cognos/APL2000 Inc	poa	Leading distributor of APL2000 products in Denmark
	Dyadic Systems Ltd.	poa	Leading distributor of Dyalog APL products in Denmark
	IBM	poa	Leading distributor of IBM APL & GraphX products in Denmark
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.
JSoftware Inc.	J on the Web online registration ...		
	J Professional (online reg.)	$895	includes manual set and one year of updates
	J Standard (online reg.)	Free	Free for download only
	Books and accessories		
	J Dictionary	$50	
	J Phrases	$50	
	J Primer	$50	
	Fractals, Visualization and J	$80	
	Concrete Math	$40	
	Exploring Math	$50	
Lescasse Consulting	APL+PC	poa	Lescasse Consulting is the exclusive APL2000 distributor in France and also distributes in Switzerland and Belgium. Call for price quotes.
	APL+Unix	poa	
	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	APLX for Windows/MacOS	499	Cross-platform APL development environment with GUI programming facilities. Interpreter modelled on APL2. Available for Windows 95/98/ME/NT/2000/XP, Mac OS 9 and Mac OS X.

	APLX Server Edition	poa	For running large multi-user APL applications on x86 Linux, RS/6000 AIX, and Windows NT/2000.
Oasis	Dyalog APL	poa	Dyadic Systems
	APL*PLUS	poa	Manugistics
	APL.68000	poa	MicroAPL Ltd
	APL2	poa	IBM
Omega	Zero	poa	A "small simple and fast" alternative to APL
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 OS/390	poa	for IBM OS/390 mainframes
	SHARP APL for UNIX	poa	for SunOS and IBM AIX
	SHARP APL for Linux	poa	for Intel Linux
Strand Software	Canada		
	All APL*PLUS products	poa	All APL*PLUS products including upgrades and educational.
	Dyadic and JSoftware products	poa	
	USA		
	Dyadic and JSoftware 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	Adaytum e.Planning	poa	Adaytum e.Planning offers a Web-based solution that combines planning, forecasting, budgeting, modelling and reporting in a single, integrated application.
APL Software\Services	APL Utilities	poa	Software: mostly .AWS for DOS, utilities for most APL interpreters. Public domain APL*Plus v10 with on-screen documentation and interactive tutorials. APL Conference Software. Books: APL user manuals for STSC, IBM, and Sharp. Request email catalog from dick.holt@juno.com.
APL Systems IDC SL	APL+Linkpro 3.0 New System	646	Database Access SQL Client for OBDC
	Upgrade to v3.0		
	from APL+Linkpro-32	280	
	from APL-Link or APL+Linkpro	474	
	GraphX (Includes ChartX)	599	
	GraphX Lite (No distribution)	340	
	SPREAD (An APL grid control)	86	
	QPLOT (x,y plotting in APL)	47	
	QWIN	215	(Runs APL+DOS functions in APL+Win)

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.
Causeway	*CausewayPro* for Dyalog/W	400	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
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	3-way TANGRAM	poa	3-way TANGRAM is a DSS-OLAP product, basically a powerful and versatile handler of multi-way tables (also known as hypercubes). It entails 46 analysis modules, including computations, cube merging, sensitivity analysis, time intelligence, free format queries, HTML and LaTeX outputs. Current version is 7.0 At the time of writing, the product is available on Dyalog APL 8.3.1 for Windows 95/98/NT. Future plans include an APL+WIN version and later a LINUX version.
CoSy	K.CoSy	$30 % yr sub	K.CoSy is a general purpose computing and programming environment constructed, all in open code, in Arthur Whitney's very high level, yet structurally transparent Array Programming Language, K , and its tightly coupled GUI. K.CoSy is an extremely productive environment in one of the most powerful and fastest of APL's progeny, and therefore, likewise, of all languages. K.CoSy provides a workspace-like interactive development environment previously impossible in K. Because of its unique open construction within the language itself, this environment is clearly competitive in a large domain with the APLs from the other vendors, for just a token cost. K.CoSy notepad nature, interactivity, and open K code vocabulary make learning Arthur Whitney's K far less daunting and far more productive that its raw console, or any external scripting method. K.CoSy being young and developing open code, lends itself to offer by subscription. Charter subscriptions are just $30 (half that for students) . (Requires separate download of K from Kx.com .)
DynArray	DynaWeb Server	poa	A web server providing web based access to applications running on the DICE interpreter from DynArray, or on an IBM mainframe running APL2.
	DynaHarry	poa	A DSS system which offers the next generation capabilities for current APLDI, IC/E and IC/1 users. It comes with ROLAP capabilities, multisystem access to a wide variety of databases and data warehouses.
	DynaLink	poa	An ODBC client interface for DICE and IBM APL2 programs.
HMW	4XTRA	poa	Networked, Windows/Unix based Front End and Middle Office Foreign Exchange and Money Market Dealing System. Scalable from 1 user to 120+.

	Inca	poa	Software Change Management System. Enables the user to co-ordinate development work from several sources, resolve clashes, promote work items for testing and configure releases to a live environment.
	Maya	poa	APL code file manager. A comprehensive suite of tools giving a multi-window IDE style interface to file based APL code. Offers features such as copying from file to file, object comparison, string search, style formatting, hot-spot editor for filed objects (including variables), etc.
	Aztec	poa	System shell for APL development. Manages real-time and batch applications across multiple platforms. Offers standardised error trapping, job scheduling, task communication and recovery/restart features.
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
Insight Systems	Causeway	poa	Leading distributor of Causeway products in Denmark
	All our old products are now either OEM'd, in the public domain, out of date, or all of the above. We'll be back!		
JAD Software	JAD SMS	poa	JAD SMS is a multi-user software management system for Dyalog APL™ based on shared, hierarchical databases. JAD SMS databases let you keep historical versions of apl items as well as attributes such as timestamp, user name and documentation. The software includes a graphical user interface as well as specialized functions for inclusion in applications. No charge for single-user version; $100/user for multiple users
Lescasse Consulting	APL+Win Monthly Training	$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	FACS	poa	EMMA-like interface to DB2 or ODBC databases
	QWIN	poa	Legacy DOS Windowing support for APL+Win
	ODBC/127	poa	IBM AP127-like ODBC Interface for APL+Win and Dyalog APL/W
Qualedi	Qualedi	$850-$5,500	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
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.
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.
Weighahead Systems	Weighahead Windows Weighing System (3WS)	poa	Recipe Weighing System for Manufacturing Industries. Pharmaceutical, Cosmetics, Foods etc. Works without keyboard or mouse. Uses Electronic Balances, Laser scanners, bar codes and label printers.
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
Andrews	Consultancy	poa	APL programming and analysis, Year-2000 legacy systems, algorithms, tree-processing.
APL Borealis Inc.	Support and Development	poa	APL Software Support and Development. Specialists since 1979 in Sharp APL, APL*Plus, APL+Win, Dyalog APL
APL Solutions Inc	Consultancy	poa	APL systems design, development, maintenance, documentation, testing and training. Providing APL solutions since 1969.
APL Systems IDC SL	Consultancy	poa	Consultancy and maintenance available by retainer or on call.
AUSCAN Software	Consultancy	poa	APL software development, training
Camacho	Consultancy	poa	Manuals; feasibility reports and estimates; analysis and programming; APL and MS Windows applications; Sharp, ISI APL, APL*PLUS, APL2/PC and other APLs spoken. Fixed price systems a speciality
Ian Clark	Consultancy	poa	Interfacing APL, VB, C/C++, ActiveX/COM. Screen design and documentation. National language porting.
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.

CoSy	Consultancy	poa	CoSy.com, Coherent Systems, provides rapid development in the K language and associated data base products, with a particular interest in quantitative (financial) modeling. A great deal can be done in a day.
David Crossley	Consultancy	poa	Experienced in large APL system developments since 1969 for PC or mainframe.
Dinosoft Oy	Consultancy	poa	Specialised in very large databases.
Dittrich & Partner	Consultancy	poa	APL programming and analysis; APL workshops and training on the job
Dyadic	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
DynArray	Consultancy	poa	DynArray offers consulting in the areas of DSS, Y2K and APL programs upgrade/conversion to modern Web enabled platforms.
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.
HMW	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Hoekstra Systems	Consultancy	poa	APL consultancy, programming, etc. Also UNIX system administration
Michael Hughes	Consultancy	poa	APL consultant with 20 years experience with all versions of APL. I can create your dynamic Web sites using the full power of APL working with Microsoft IIS (Internet Information Service) on Windows NT or 2000. I also undertake System design, Programming and Maintenance on all platforms, particularly MS Windows.
INFOSTROY	Consultancy	poa, competitive	Broad experience in various APL platforms. Special skills and knowledge in developing complex applications for investment, financial and construction markets. Implementation of hybrid solutions based on APL, Delphi, C++, VBA, SQL servers.
Insight Systems	Consultancy	poa	We have experience with just about every APL system and platform in common use during the last 20 years, from SHARP APL under MVS or Linux to APL+Win and in particular Dyalog APL under Windows 9x, NT or 2000. If you have decisions to take about adapting your APL application to take advantage of emerging technologies, or would like your strategy reviewed, give us a call. We have extensive experience in all areas of APL development, from legacy systems, up, down and sideways migrations, to the development and support of shrink wrapped solutions based on APL. Even if we don't have time to do the work ourselves, we will know where to find someone who is an expert in your version of APL and your application area, on your continent.
JAD Software	Consultancy	poa	Systems design and development, project management, technical manuals, financial and actuarial expertise in APL.
KJK	Consultancy and software development	poa	APL-based data management: conversions, ad hoc-analyzing tools, well-interfaced methods for defining, processing and browsing of multi-dimentional reports. Rapid custom software development based on proven modular toolset approach.
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.
Lucas Solutions	Consultancy	poa	Rates depend on task and location.
George MacLeod	Consultancy	poa	Design and programming of new APL applications. Enhancing and maintaining existing APL applications. Porting existing APL applications from one APL system to another. Supporting users of APL applications. Experienced on both mainframe, UNIX and PC APL interpreters.
Mackay Kinloch Ltd	Consultancy	poa	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
MasterWork Software	Consultancy	poa	Consulting and J programming for econometrics and statistics in public policy, health and food industries.
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, 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)
Omega Computing	Consultancy	poa	APL consultancy, programming, etc.
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.
Resources & Results	Consultancy	poa	Knowledge management company builds decision support, data warehouse, data mining and strategic planning systems for CFO's, CEO's, and senior management, using our Rapid Application Development (RAD) methods and tools. Extensive experience in large-scale system development and ad hoc executive support for Fortune 500 clients.
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.
SovAPL	Consultancy	poa	Offshore APL development service.
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.
Weighahead Systems	Consultancy	poa	Specialising in industrial systems. Links to PLCs, laser scanners, bar codes, weigh scales, label printes etc. Also programmable hand held scanners.
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.
APL Borealis Inc.	APL Training	poa	Hands-on courses in Introductory, Intermediate, Advanced and Windows APL. Courses are customized and flexible, and may be delivered on-site, with strong emphasis on methods for efficient and maintainable APL systems development.
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.

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 Germany e.V.	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			
SE APL Users Grp	Atlanta, Georgia	SEAPL Newsletter	Quarterly meetings	$10
SovAPL	Moscow, Russia	-	Seminars and Annual Meeting	
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	Stellinger Weg 19, 20255 Hamburg, Germany. Tel: +49 40 40170951 Fax: +49 40 40170952. Email: info@adapta.de
Adaptable Systems	Lois & Richard Hill	49 First Street, Black Rock 3193, Australia. Tel: +61 3 9589 5578 Fax: +61 3 9589 3220 Email: hillrj@melbpc.org.au
Adaytum Limited	Douglas Rowley	Castlegate, Tower Hill, BRISTOL BS12 0JA. Tel: 0117 921 5555 Fax: 0117 922 7749. Email:sales@adaytum.co.uk
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 Email: ADWilson@kencomp.net
APL-385	Adrian Smith	Brook House, Gilling East, York YO62 4JJ, UK. Tel: 01439-788385 Email: apl385@compuserve.com
APL2000 (Europe)	Fred Honea	see APL Systems IDC SL.
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 Borealis Inc.	Richard Procter	381 Manor Road East, Toronto, Ontario M4S 1S7, Canada. Tel: (416) 457-7828. Fax: (416) 482-6582 Email: info@aplborealis.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 Germany e.V.	Dieter Lattermann	Rheinstraße 23, D-69190 Walldorf, Germany. Tel: +49 6227-63469 Compuserve: 100332,1461
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: aplsi@starpower.net
APL Systems IDC SL	Fred Honea	Alfredo Marquerie, 12 - 2 F, 28034 Madrid, Spain. Tel: +34 91 730 7008 (Office) +34 60 680 5949 (Mobile). Fax: +1 775 743 6131. Email: uksales@apl2000.net
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	Rooinberg 72, B-2570 Duffel, Belgium. Tel: +32 15 31 47 24
Beautiful Systems, Inc.	Jim Goff	308 Old York Road, Suite 5, Jenkintown, PA 19046, USA Tel: +1 (215) 886-2636; Fax: +1 (215) 886-4888 Email: BeautifulSystems@goffs.net
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. email: acam@tesco.net
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS, UK. Tel: 01252-874697 Email: ray_cannon@compuserve.com
Causeway Graphical Systems Ltd	Adrian Smith	The Maltings, Castlegate, MALTON, North Yorks YO17 7DP, UK. Tel: 01653-696760 Fax: 01653-697719 Email: adrian@causeway.co.uk
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB, UK. Tel: 020 7404 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
Ian Clark	Ian Clark	1205 Deer Creek Drive, Plainsboro, NJ 08536, USA. Tel: +1 609 716 8832 Email: earthspot2000@hotmail.com
CODEWORK	Mauro Guazzo	Corso Cairoli 32, 10123 Torino, Italy. Tel: +39 11 885168 Fax: +39 11 812 2652 Email: codework@codework-it.com
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.
CoSy.com	Bob Armstrong	42 Peck Slip #4B, New York, NY 10038-1725, USA. Tel: +1 212-285-1864 Fax: +1 212-285-1864. E-mail: bob@CoSy.com.
David Crossley	David Crossley	187 Le Tour du Pont, 84210 ST DIDIER, France. Tel: +33.4.90.66.08.87 Email: crossley@au-village.com

Danish User Group	Helene Boesen	c/o Insight Systems ApS, Nordre Strandvej 119G, Hellebæk, Denmark
Dinosoft Oy	Pertti Kalliojärvi	Lönnrotinkatu 21C, 00120 Helsinki, FINLAND. Tel: +358 9 70028820 Fax: +358 9 70028824 Email: dinosoft@dinosoft.fi
Dittrich & Partner Consulting GmbH	Axel Holzmüller	Kieler Strasse 17, D-42697 Solingen, Germany. Tel: +49 212-260 660 Fax: +49 212-260 6666; Email: info@dpc.de
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
DynArray Corporation	Dr James Brown	16360 Monterey Rd. Suite 260, Morgan Hill, CA 95037, USA Tel: +1 (408)-782-6648 Fax: +1 (408)-782-6627 Email:info@DynArray.com
Evestic AB	Olle Evero	Berteliusvagen 12A, S-146 38 Tullinge, Sweden Tel & Fax: +46 778 4410 Email: olle.evero@mail.com
FinnAPL	Olli Paavola	Suomen APL-Yhdistys RY, FinnAPL RF, PL 1005, 00101 Helsinki 10, Finland Email: olli.paavola@pyr.fi
First Derivative Analytics Ltd.	Ken Chakahwata	114 Lemsford Lane, Welwyn Garden City, Herts AL8 6YP, UK 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
HMW Computing	Chris Hogan	Hamilton House, 1 Temple Avenue, London EC4Y 0HA, UK. Tel: 0870-1010-469; Email:HMW@4xtra.com
Hoekstra Systems Ltd	Bob Hoekstra	Dominique, Salisbury Road, Woking, Surrey, GU22 7UR, UK. Tel: 01483-771028. Fax: 01483-837324 Email: Bob.Hoekstra@HoekstraSystems.ltd.uk
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics. LE16 8QL, UK. Tel: 01536-770998 Email: Michael@Hughes.uk.com
I-APL Ltd	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. Email: 100612.1057@compuserve.com
IBM APL Products	Nancy Wheeler	APL Products, IBM Silicon Valley Lab, Dept H36/F40, 555 Bailey Avenue, San Jose CA 95141, USA. Tel: +1 (408) 463-APL2 [+1 (408) 463-2752] Fax: +1 (408) 463-4488 Email: APL2@vnet.ibm.com
INFOSTROY	Alexey Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812 325-9797 Fax:+7 812 311-2184 Email:aim@infostroy.ru
Insight Systems ApS	Helene Boesen	Nordre Strandvej 119G, DK-3150 Hellebæk, Denmark Tel:+45 70 26 13 26 Fax: +45 70 26 13 25 Email: info@insight.dk
JAD Software	David Crossley	175 East 96th St., Apt. 17G, New York, NY 10128 Country: USA Tel: +1 (212) 369-6713 Fax: +1 (212) 761-0124 Email: jadsms@usa.net
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	Hochsteingasse 13/26, 8010 Graz, Austria. Tel:0043 (0)316 91 42 51 Mobile: 0043 (0)699 1 91 42 51 2. Email: joachim.hoffmann@gmx.at
JSoftware Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9. Tel: +1 (952) 470-7345 Fax: +1 (952) 470-9202 Email: info@jsoftware.com
KJK-tieto Oy	Kimmo Kekäläinen	Merikasarminkatu 10 B 56,00160 Helsinki, Finland. Tel: +358 50 55 27 207; Email: Kimmo.Kekalainen@pp.htv.fi
Phil Last Ltd	Phil Last	146 Crossbrook Street, Cheshunt, Herts, EN8 8JY, UK. Tel: 01992-633807 Fax: 0121-359 0375 Email: phil.last@net.ntl.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	Walter G. Fil	203 N. LaSalle Street, Suite 2100, Chicago, Illinois 60601, USA. Tel:+1 (800) 546 4621 E-mail: lingo-allegro@visto.com
Lucas Solutions	Jim Lucas	Stubbedamsvej 9C, 3.tv., 3000 Helsingør, Denmark Tel: +45 49 26 52 42. Email: jel@danbbs.dk
Mackay Kinloch Ltd	Alastair Kinloch	519 Webster's Land, Edinburgh EH1 2RX, Scotland, UK. Tel: +44 (0)131 228 5235 Email: alastair.kinloch@btinternet.com

George MacLeod	George MacLeod	37 Newhouse Rd, Bovingdon, Herts, HP3 0EJ, UK. Tel: 01442-831385 Fax: 01442-831388 Email: george.macleod@ntlworld.com
MasterWork Software Ltd	Fraser Jackson	PO Box 56-036, Tawa, Wellington, New Zealand. Tel: +64 (4) 232-4440 Fax: +64 (4) 232-4452 Email: fraser.jackson@xtra.co.nz
Mercia Software Ltd.	Gareth Brentnall	Mercia House, Ashted Lock, Aston Science Park, Birmingham, B7 4AZ, UK. Tel: 0121-359 5096. Fax: 0121-359 0375
MicroAPL Ltd.	Richard Nabavi	The Roller Mill, Mill Lane, Uckfield, E.Sussex TN22 5AA Tel: 01825 768050. Fax: 01825 749472 Email: MicroAPL@microapl.demon.co.uk
Milinta Inc.	Dan Baronet	Contact Dan Baronet at danb@milinta.com
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.	Louis Rijkse	Lekstraat 4, 3433 ZB Nieuwegein, Holland. Tel: +31 30 60 66 336 Fax: +31 30 60 65 844 Email: rijkse@oasis.nl
Omega Computing Inc	Alan Graham, Andrew Chou	3 Columbus Avenue, Edison, NJ 08817, USA. Tel: +1 (732) 985 9519 Email: alangraham@mindspring.com
Optima Systems Ltd	Paul Grosvenor	1st Floor, 10 Raleigh Court, Priestley Way, Crawley, West Sussex, RH10 2PD, UK. Tel: 01293 562700 Fax: 01293 562699 Email:mailbox@optima-systems.co.uk
Qualedi Inc.	Nicole Schless Georges Brigham	121 West Main Street, Milford, CT 06460, USA. Tel: +1 (203) 874-4334 Fax: +1 (203) 876-9083. Email: sales@qualedi.com; info@qualedi.com
RadSys Technologies AB	Randolph Schrab	Lovsangarv. 18, S-756 52 Uppsala, Sweden. Tel: +46 18 32 41 53 Fax: +46 708 1996 11 Email: randolph.schrab@radsys.se
Renaissance Data Systems	Ed Shaw	P.O. Box 313, Newtown, CT 06470, USA. Tel: +1 (203) 270-9729 Email: rendata@aplbooks.cnchost.com or orders@aplbooks.cnchost.com
Resources & Results	Frank Rhodes	2438 W. Northgate Dr., Irving, TX 75062, USA. Tel: +1 972 523 5117 Email: frank@decision-support.net
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, 00149-Roma Trullo, Italy Email: mario.sacco@tin.it
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 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@interlog.com
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	Benoit Paquin	Soliton Denmark, Havsgårdsvej 4, 2900 Hellerup, Denmark Email: sales@soliton.com
SovAPL Russian Chapter of SIGAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region 249020, Russia Tel: +7(08439)47109 Fax: +1 (530) 6885510 Email:askom@obninsk.com
Strand Software Inc	Anne Faust	19235 Covington Court, Shorewood MN 55331 USA Tel: +1 (952) 470-7345 Fax: +1 (952) 470-9202 Email: info@strandsoft.com
Rex Swain	Rex Swain	8 South Street, Washington, CT 06793 USA. Tel: +1 (860) 868-0131 Fax: +1 (860) 868-9970 Email: rex@rexswain.com
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@ifi.unizh.ch
Sykes Systems Inc	Roy Sykes Jr	4649 Willens Ave., Woodland Hills, CA 91364, USA Tel: +1 (818) 222-2759 Fax: +1 (818) 222-9250

Toronto SIG	Richard Procter	PO Box 55, Adelaide St. Post Office, Toronto Ontario M5C 2H8, Canada Email: info@torontoapl.org
Weighahead Systems	Philip Bulmer	Camberley House, 1 Portesbery Road, Camberley, GU15 3RB, UK. Tel +44 1276 20789 Email: sales@weighahead.com
Stephen Wynn		8 Clarence Gardens, Brighton, Sussex BN1 2EG, UK. Tel: 01273-327238 Email: centre@cwcom.net
Zark Incorporated	Gary A. Bergquist	23 Ketchbrook Lane, Ellington CT 06029, USA. Tel: +1 (860) 872-7806

FTP SITES

IBM APL2	ftp.software.ibm.com/ps/products/apl2
Waterloo Archive	archive.uwaterloo.ca/ftparch/languages/apl
APL-to-ASCII	archive.uwaterloo.ca/languages/apl/workspaces/aplascii

WORLD WIDE WEB SITES

ACM SigAPL	www.acm.org/sigapl/
Adapta Software	www.adapta.de/
Adaptable Systems	www.assuredsystems.com.au/
Adaytum Limited	www.adaytum.com/
AFAPL	www.afapl.asso.fr/ (Journal available on line)
APL2000	www.APL2000.com/
APL-385	www.demon.co.uk/apl385/
APL Journal, Germany	www.rhombos.de/apljourn.htm
APL Systems IDC SL	www.apl2000.net
AUSCAN	www.interlog.com/~rjp/auscan/
Eke van Batenburg	wwwbio.LeidenUniv.nl/~Batenburg/index.html
Capital PC User Group	http://cpcug.org/
Causeway	www.causeway.co.uk/
CODEWORK	www.codework-it.com/tangram/eng/
CoSy (Bob Armstrong)	CoSy.com/
Dinosoft Oy	www.dinosoft.fi/
Dittrich & Partner	www.dpc.de; www.apl-online.de
DMOZ - Open Directory	http://dmoz.org/Computers/Programming/Languages/APL/
Dyadic Systems Ltd	www.dyadic.com/
DynArray	www.dynarray.com/
FinnAPL	www.pyr.fi/apl/
Godin London Inc	www.godin.com/
Houben (IQL)	www.apl.olap.club.tip.nl
Hoekstra Systems	www.HoekstraSystems.ltd.uk/
IBM APL2	www.ibm.com/software/ad/apl
Infostroy	www.infostroy.ru
Insight Systems ApS	www.insight.dk/
Japan APL Association	www.naska.co.jp/JAPLA/
JSoftware Inc	www.jsoftware.com/
KJK-tieto Oy	www.kjk-tieto.com
Lescasse Consulting	www.lescasse.com/
Lingo Allegro USA Inc	www.lingo.com/
Mackay Kinloch	http://mackaykinloch.ltd.uk/

MicroAPL Ltd	www.microapl.co.uk/apl
Milinta Inc	www.milinta.com
Oasis b.v.	www.oasis.nl/
Optima Systems Ltd	www.optima-systems.co.uk
Qualedi, Inc.	www.qualedi.com
Renaissance Data	www.aplbooks.cnchost.com/
Resources & Results	www.decision-support.net
RE Time Tracker Oy	www.rett.fi/
The Rochester Group Inc.	www.rochgrp.com/
Shepp & Associates	www.digitravel.com/
SigAPL	www.acm.org/sigapl/
Soliton	www.soliton.com/
Snake Island Research Inc.	www.snakeisland.com
Strand Software Inc.	www.strandsoft.com/
Rex Swain	www.rexswain.com/
Toronto SIG (for Toolkit)	www.torontoapl.org/
Jim Weigang	www.chilton.com/~jimw/
Weighahead Systems	www.weighahead.com

APL BOREALIS

Software * Support * Training

APL **Software** Development and Support All platforms Specialists since 1973	***APL+Win*** and ***Dyalog APL*** Sales and Support	***APL*** **Training** - Customized Courses Introductory to Advanced levels *APL* system design Windows *APL*

Canada's *APL* Product and Service Provider

www.aplborealis.com

Toronto Canada tel: 416.457.7828 info@aplborealis.com

ZARK Newsletter Extracts

introduced by Jon Sandles

Utility Corner: Cross Tabulations

(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.)

Suppose you have a "database" of employee information. For each of the 2000 employees, you have four items of information: age, state of residence, type of employee, and bonus amount received last year. Here are a few records:

AGE	*STATE*	*TYPE*	*BONUS*
26	*FL*	*E*	200
39	*MI*	*S*	550
31	*NY*	*O*	720
52	*NY*	*S*	375

The information is stored in your APL workspace in four simple variables: *AGE*, *STATE*, *TYPE*, and *BONUS*. *AGE* and *BONUS* are 2000-element numeric vectors; *STATE* is a 2000-row, 2-column character matrix; and *TYPE* is a 2000-element character vector.

In last issue's *Limbering Up* column, we asked you to summarise this mass of information by grouping the employees by age, state, and type. We gave you the following function to complete:

```
      ∇ R←CROSSTAB
[1]    ⍝ Summarises employee information
[2]    ⍝ stored in workspace globals AGE,
[3]    ⍝ STATE, TYPE, and BONUS.  AGE and
[4]    ⍝ BONUS are numeric vectors, STATE
[5]    ⍝ is a two-column character matrix,
[6]    ⍝ and TYPE is a character vector.
[7]    ⍝ Their elements/rows are in 1-to1
[8]    ⍝ correspondence.  R is a four-
[9]    ⍝ dimensional array whose shape is
[10]   ⍝ (5 10 3 2):
```

```
[11]  ⍝
[12]  ⍝  R[I;;;]  The Ith age range:
[13]  ⍝       I=1;   0≤AGE≤29
[14]  ⍝       I=2:  30≤AGE≤39
[15]  ⍝       I=3:  40≤AGE≤49
[16]  ⍝       I=4:  50≤AGE≤64
[17]  ⍝       I=5:  65≤AGE
[18]  ⍝  R[;J;;]  The Jth state.  The glo-
[19]  ⍝       bal ALLSTATES is a 10-row,
[20]  ⍝       2-column character matrix of
[21]  ⍝       the codes for all ten states
[22]  ⍝       in which the employees reside.
[23]  ⍝  R[;;K;]  The type 'EOS'[K]
[24]  ⍝  R[;;;1]  The employee count.
[25]  ⍝  R[;;;2]  The total bonus amount.
    ∇
```

The idea behind a function like *CROSSTAB* is to process the database once, summarising its information into a single multidimensional array. All subsequent analysis can then be efficiently performed on the array, rather than reprocessing the thousands (or millions) of records.

The typical APL approach to solving multidimensional cross-tabulation problems like the one above is to use outer and inner products. For example, you can use ∘.= to get the number of employees by type:

```
        +⌿TYPE∘.='EOS'
```

It's a small step to get the bonus amount by type:

```
        BONUS+.×TYPE∘.='EOS'
```

By laminating a row of 1s onto *BONUS*, you can get both employee count and bonus amount with one expression:

```
        (1,[.5]BONUS)+.×TYPE∘.='EOS'
```

Likewise, you can get employee count and bonus amount by state:

```
        (1,[.5]BONUS)+.×STATE∧.=⍉ALLSTATES
```

How do we put the two together to get employee count and bonus amount by state? Unfortunately, the inner/outer product approach gets *much* harder to formulate as you go beyond two dimensions. Here's a painful, but successful, attempt:

```
(((10 3,⍴TYPE)⍴3⌿ALLSTATES∧.=⍉STATE)∧(10 3,⍴TYPE)⍴'EOS'∘.= TYPE
)+.×1,[1.5]BONUS
```

This expression requires the reader to do a lot of head-scratching; and it requires the computer to do a lot of work. Instead of getting out the crow-bar to pry the final dimension (age) into the expression, let's look for a different approach.

One possibility is to use nesting:

```
↑((↓ALLSTATES∧.=⍉STATE)∘.∧'EOS'=⊂TYPE)+.×¨⊂1,[1.5]BONUS
```

(Replace the APL*PLUS split ↓ and mix ↑ function by ⊂[2] and ⊃[2] if using APL2.)

It is debatable whether this expression requires more or less head-scratching. Moreover, it is *not* more efficient than the previous expressions. All the same comparisons are being made. However, it *does* lend itself to be extended to higher dimensions. (Bet you can hardly wait!)

```
↑((↓<⍀29 39 49 64 99∘.≥AGE) ∘.∧(↓ALLSTATES∧.=⍉STATE) ∘.∧'EOS'=⊂
TYPE) +.×¨⊂1,[1.5]BONUS
```

This expression solves our problem, but is extremely inefficient. Let's formulate a different approach by working backwards from the desired result. Where do each of the original records affect the eventual result?

AGE/ ind.	*STATE/* ind.	*TYPE/* ind.	*Result* ind.	*BONUS*
26/1	*FL*/5	*E*/1	13	300
39/2	*MI*/8	*S*/3	54	550
31/2	*NY*/3	*O*/2	38	720
52/4	*NY*/3	*S*/3	99	375

In this table, we've redisplayed the four sample records from above. New each *AGE*, *STATE*, and *TYPE* value, we've included the index of the class implied by that value. For example, look at the last record. Age 52 is in the 4th age class (50 to 64); *NY* is in the 3rd state class (i.e. *ALLSTATES*[3;]='*NY*'); and *S* is the 3rd employee type (*E*=1, *O*=2, *S*=3).

In an array of 5 (age) by 10 (state) by 3 (type) possibilities, the cell affected this last record is the one in the 4th plane, 3rd row, and 3rd column, or the 99th cell out of 150:

```
      1+5 10 3⊥4 3 3-1
99
      1+30 3 1+.×4 3 3-1
99
```

By translating the age, state, and type values into class indices, and then combining them into a single "result index", the multi-dimensional problem has been reduced into a one-dimensional problem. You have all the information you need to solve the problem in the last two columns of the table above.

Accumulate the *BONUS* amounts and record counts by each distinct result index; assign the accumulated amounts and counts into their corresponding positions of two 150-element vectors (or a 150 by 2 matrix); and reshape the result into the desired 4 dimensions.

The accumulation logic is the standard (and efficient) sort, shift, and compare logic that's been around for eons. Here's the finished code:

```
      ∇ R←CROSSTAB;AI;B;C;G;L;RI;SI;SRI;TI
 [1]   ⍝ Summarises employee information
  :        :        :        :
 [26]  ⍝ AGE class index:
 [27]   AI←0 30 40 50 65 LIOTAI AGE
 [28]  ⍝ STATE class index:
 [29]   SI←ALLSTATES CMIOTA STATE
 [30]  ⍝ TYPE class index:
 [31]   TI←'EOS'⍳TYPE
 [32]  ⍝ Result index:
 [33]   RI←(30×AI-1)+(3×SI-1)+TI
 [34]  ⍝ Sorted result indices:
 [35]   SRI←RI[G←⍋RI]
 [36]  ⍝ Flag last of each like index:
 [37]   L←SRI≠1↓SRI,¯1
 [38]  ⍝ Distinct result indices:
 [39]   SRI←L/SRI
 [40]  ⍝ Cumulative employee counts
 [41]   C←L/⍳⍴L
 [42]  ⍝ Employee counts
 [43]   C←C-¯1↓0,C
 [44]  ⍝ Cumulative bonus amounts:
 [45]   B←L/+\BONUS[G]
 [46]  ⍝ Bonus amounts:
 [47]   B←B-¯1↓0,B
 [48]  ⍝ Construct result; insert; reshape:
 [49]   R←150 2⍴0
 [50]   R[SRI;]←C,[1.5]B
 [51]   R←5 10 3 2⍴R
      ∇
```

Notice the use of *LIOTAI* (lower limit iota, inclusive), presented in the previous newsletter, and the of *CMIOTA* (character matrix iota), presented several issues back, to translate the age and state values to indices.

The logic in the above *CROSSTAB* function has been generalised and included in a utility function named *PLUSRED*. Here's how you'd write *CROSSTAB* using *PLUSRED*:

```
      ∇ R←CROSSTAB;AI;SI;TI
 [1]   ⍝ Summarises employee information
  :        :        :        :
 [26]  ⍝ AGE class index:
 [27]   AI←0 30 40 50 65 LIOTAI AGE
 [28]  ⍝ STATE class index:
 [29]   SI←ALLSTATES CMIOTA STATE
 [30]  ⍝ TYPE class index:
 [31]   TI←'EOS'⍳TYPE
 [32]  ⍝ The data to summarise (incl freq):
 [33]   R←1,[0.5]BONUS
 [34]   R←5 10 3 AI SI TI PLUSRED R
      ∇
```

The left argument of *PLUSRED* is a nested argument. (A non-nested version of *PLUSRED* can be found in the Zark Library of Utility Functions.) The first three items are the class sizes. There are 5 age classes, 10 state classes, and 3 type classes. The next three items are the respective vectors of class indices. The right argument is the array to be summarised; its last dimension is "reduced," i.e. replaced by the class sizes. In this case, if the argument has shape `2 2000`, the result has shape `2 5 10 3`. If the shape of the argument was instead `2000 2`, we would want to "reduce" the first dimension (the 2000), and get back a result with shape `5 10 3 2`. To specify a reduction dimension other than the last, include its index at the end of *PLUSRED*'s left argument, like so:

```
 [33]   R←1,[1.5]BONUS
 [34]   R←5 10 3 AI SI TI 1 PLUSRED R
```

Here's the *PLUSRED* utility function:

```
     ∇ R←L PLUSRED ARRAY;CUM;DIM;DSHAPE;G;I;LAST;M;N;RANK;RRI;S;U
  [1]   ⍝ No. of ways for N-way reduction:
  [2]    N←⌊(×/⍴L)÷2
  [3]   ⍝ Which dimension to be "reduced"?
  [4]    DIM←1↑((N+N)↓L),¯1↑⍳RANK←⍴⍴ARRAY
  [5]   ⍝ Get replacement shape from left arg:
  [6]    DSHAPE←N⍴L
  [7]   ⍝ Initial "raveled reslt inds":
  [8]    I←⎕IO
  [9]   ⍝ Index from ⍳DSHAPE[I] to cause
  [10]  ⍝ index err if invalid indices:
  [11]   RRI←(⍳DSHAPE[I])[(N+I)⊃L]
  [12]  ⍝ Continue computing RRI (N loops
  [13]  ⍝ for N-way reduction):
  [14]  L1:→((N+⎕IO)≤I←I+1)⍴L2
  [15]   RRI←(⍳DSHAPE[I])[(N+I)⊃L]+DSHAPE[I]×RRI-⎕IO
  [16]   →L1
  [17]  ⍝ Determine unique elements of RRI:
  [18]  L2:S←RRI[G←⍋RRI]
  [19]   U←(LAST←S≠1↓S,¯1)/S
  [20]  ⍝ Reorder ARRAY to conform with S:
  [21]   M←DIM-⎕IO
  [22]   ARRAY←⍎'ARRAY[',(M⍴';'),'G',((RANK-M+1)⍴';'),']'
  [23]  ⍝ Perform partitioned reduction:
  [24]   CUM←LAST/[DIM]+\[DIM]ARRAY
  [25]   CUM←CUM-(⍴CUM)↑0,[DIM]CUM
  [26]  ⍝ Initialise result. Fill with 0s
  [27]  ⍝ Ravel the DSHAPE dim.s:
  [28]   R←((-M)⌽(×/DSHAPE),1↓M⌽⍴ARRAY)⍴0
  [29]  ⍝ Insert result of part. reduction:
  [30]   ⍎'R[',(M⍴';'),'U',((RANK-M+1)⍴';'),']←CUM'
  [31]  ⍝ Reshape to desired shape:
  [32]   R←((-M)⌽DSHAPE,1↓M⌽⍴ARRAY)⍴R
      ∇
```

In case you missed the last few issues, here are the *LIOTAI* and *CMIOTA* utility functions:

```
∇ R←LOWER LIOTAI VALS;G;I;M
   ⍝ Returns the index of the value of
  ⍝ the vector LOWER that is the
  ⍝ nearest lower bound (inclusive)
  ⍝ for each element of the vector
  ⍝ VALS. Values in LOWER need not be
  ⍝ ascending or distinct. ⎕IO+⍴LOWER
  ⍝ is returned for elements in VALS
  ⍝ above the smallest value in LOWER
  ⍝
  ⍝ Combine and sort up to use 1st val
  ⍝ in LOWER if vals not distinct:
   R←G←⍋VALS,LOWER
  ⍝ Flag elts from LOWER in grade vec:
   I←(G≥⎕IO+⍴VALS)/⍳⍴G
  ⍝ Inds into LOWER, in sorted order,
  ⍝ using "one greater" for values
  ⍝ above largest in LOWER:
   M←(G[I]-⍴VALS),⎕IO+⍴LOWER
  ⍝ Replicate inds for values in
  ⍝ corresp. ranges:
   R[G]←((I,⎕IO+⍴G)-⎕IO,I)/M
   R←(-⍴LOWER)↓R ⍝ Discard LOWER inds
∇
```

```
∇ R←A CMIOTA1 B;G;J;K;N;S;T;Z
  ⍝ Returns row indices of character
  ⍝ matrix A where each row of B is
  ⍝ found. Like dyadic ⍳, CMIOTA re-
  ⍝ turns one greater than the largest
  ⍝ index in A for rows of B not found.
  ⍝
  ⍝ Index returnd if not found:
   Z←⎕IO+1↑⍴A
   N←1↑⍴B ⍝ No. rows in B
  ⍝ Stick ¯1↑⎕AV as new row of A:
   A←A,[⎕IO]⎕AV[255+⎕IO]
   G←⎕AV⍋B,[⎕IO]A ⍝ Grade vec
  ⍝ Which ones come from B (∧/K∊⍳N):
   K←(S←G<N+⎕IO)/G
  ⍝ The relative ind of the next A row:
   J←⎕IO+S/+\T←~S
   R←J←(T/G)[J]-N ⍝ The absolute A inds
   R[K]←J ⍝ ...in the orig order of B
  ⍝ Use the "not-found" value for those
  ⍝ not matching:
   R[(∨/B≠A[R;])/⍳N]←Z
∇
```

Limbering Up: Elvis Elbows

(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.)

Suppose you have a database of 9000 names, and you need to locate "Schneider, K." Or is it "Schnyder, K."? Or maybe "Snyder, K."? Heck, you're not sure exactly what the name is. But you do know approximately what it sounds like. What do you do?

Perhaps the U.S. government can come to the rescue. During the make-work mindset of the 1930s, as part of the massive WPA (Works Progress Administration) program, the government decided to put people to work re-indexing past censuses. In particular, they set out to re-index the U.S. federal Censuses of 1880, 1900, and 1910 by the way names *sounded,* rather than the way they were spelled.

In this way, if names were mis-spelled slightly by the census takers, a name could still be located if you knew approximately how to pronounce the name. It turns out that this phonetic indexing has been invaluable to folks doing genealogical searches.

But how did they do it? The scheme devised by the government was to treat all letters sounding approximately the same as if they are the same. For example, DD and T are the same; B and P are the same; M and N are the same; and so on. The term "Soundex" was coined to refer to this scheme. (The term "Miracode" is sometimes used too.)

Your task is to implement Soundex. In particular, write either or both of the functions *SOUNDEXV* and *SOUNDEXM*. The first takes a single character vector name as its argument and returns the Soundex code for the name. The second takes a character matrix of names, one per row, as its argument and returns a vector of Soundex codes, one per name. The functions should be as *fast* as possible.

Here are the rules for converting a name to its Soundex code:

1. Delete all blanks and nonletters from the name (Van Owen → VanOwen).
2. Convert lowercase to uppercase (VanOwen → VANOWEN).

3. Set aside the first letter (V).
4. Translate all letters, including the first, to digits between 0 and 6 via:

```
 0: A,E,H,I,O,U,W,Y
 1: B,F,P,V
 2: C,G,J,K,Q,S,X,Z
 3: D,T
 4: L
 5: M,N
 6: R

VANOWEN → 1 0 5 0 0 0 5
```

5. Eliminate contiguous duplicates (`1 0 5 0 0 0 5 → 1 0 5 0 5`).
6. Eliminate the first letter/digit (`1 0 5 0 5 → 0 5 0 5`).
7. Squeeze out all 0s (`0 5 0 5 → 5 5`).
8. Grab the first three digits, padding with 0s, and stick the first letter, from step 3, on the front (`V550`). To get a convenient, integer result, we'll deviate from strict Soundex conventions and convert the first letter to a number, its index into the alphabet (`V550 → 22550`).

Test your functions on the following:

Name	*Soundex Code*
Eberhard	5166
Elvis	5412
Kelly	11400
Schaefer	19160
Van Owen	22550

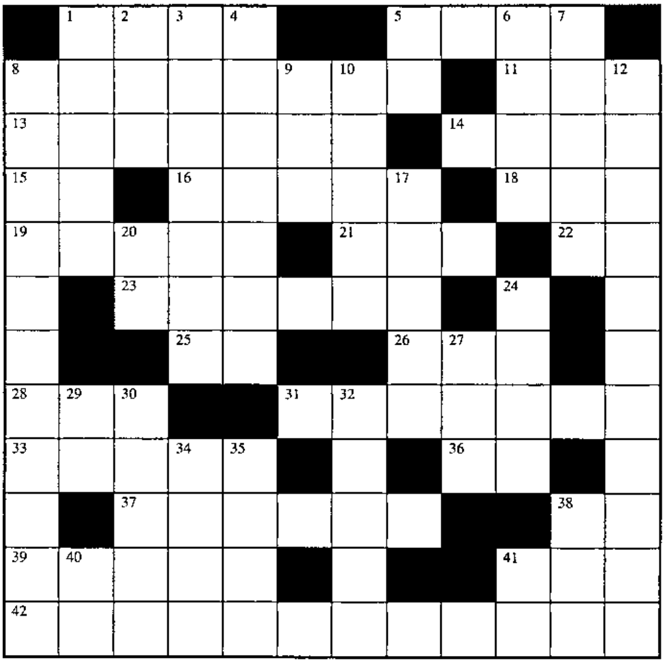
Send your solutions to:

Vector Production
Brook House, Gilling East, York YO62 4JJ
UK email apl385@compuserve.com

The notable functions and their authors' names will be printed in the next issue of *Vector*. Good luck and happy limbering.

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

Crossword



Across

1. The lesser of M or SM
5. Branch to line L: if...
8. A matrix with N rows of the vector A
11. (A,N)[A<N] for scalars A,N in ⎕IO=0
13. B=TSD exactly (as if ⎕CT=0)
14. V[2]-V[1] in origin 1 where 2=ρV
15. A wild APL party
16. (J=I)∨(J=K) for scalars I, J, K
18. The remainder of E when divided by B
19. 2 if S<0, 1 if S=0. ÷2 if S>0
21. (L≠0)/L
22. ⎕__ tells functions from variables
23. (1↓EXPR),(1↑EXPR)

25. ⎕___ lists your functions or variables
26. (×/ρVT)ρVT
28. M formatted to R decimal places
31. The product of the rows of N, summed, times A
33. Where am I?
36. −/5 3⋆2
37. −⍳N in origin 1
38. E×¯1
39. ⍳TW in random order
41. Flag the leading 1s in Boolean C
42. The element in vector RATES corresponding to the largest element in vector V

Down

1. MN∊G5 for scalar G5
2. The integral portion of the elements of array B, as a vector
3. S↑(−J)×N[⌽⍳ρN]
4. (⎕IO+ρSEL)>SEL⍳MAT
5. Branch to line A:
6. 1 if A>B, 0 if A=B, ¯1 if A<B
7. +\(|1↑N[⍒(N×N)⋆0.5])ρ1 in origin 1
8. Matrix TR, without its all-0 columns
9. What's happening?
10. (ρD)+(ρLP) for vectors D, LP
12. NVEC in ascending order
17. ((K≠1)>K∊R)/K
20. The mathematical constant e
24. The product of T and !3, rounded up to the nearest integer
27. +/×\¯1↓1,⌽V for V a vector
29. W formatted to ⎕PP siginificant digits
30. One random number from MS+1 to ML+T, in origin 1
32. (ρ,MV)[⎕IO]
34. TE×I, without multiplying, for scalar TE and Boolean vector I
35. −WS−D
38. Alternating sum of V
40. ⎕___ tells howmuch room remains
41. All

Solution to Crossword in 18.4

÷	1	0	0	■	V	~	B	T	■	■	(
/	■	=	■	(	⍋	S	,	V	)	⍳	1
M	←	1	∊	V	E	∨	0	∊	C	■	↓
T	■	↑	■	⍳	■	G	⌈	J	L	■	⍴
■	1	0	⌽	V	L	>	R	■	E	⌽	M
(	2	⍴	N	)	↑	0	■	V	A	⍳	A
■	,	A	T	=	K	■	⌊	/	R	N	T
P	I	■	/	⍳	⍴	■	.	1	■	■	)
|	B	■	N	⍴	3	■	5	■	■	⍴	⍴
9	■	B	■	V	■	A	+	.	×	L	M
■	×	\	■	■	∧	\	V	■	⍴	,	A
+	/	2	≤	+	/	S	S	⍷	M	A	T

J-ottings 33: How to Tell a Fib in lots of Different Ways

by Norman Thomson

Michel Dumontier's article in Vector vol. 18 on the "beastly number" 666 came as quite a Revelation to me, but I suppose that was always the way it was intended to be! The Fibonacci series and its squares are a bit like it in that once you start looking for them, they turn up all over the place! The Fibonacci series is well known for its application in models of natural phenomena like the arrangement of sunflower seed-heads, and the population growth of rabbit colonies. Readers no doubt know well how this series is extended by concatenating the sum of the last two items, which is an imperfect way of saying in English what a fully accurate description in J says a few lines below

Assigning the first fourteen terms to a variable f is sufficient to observe the evolving patterns. Because this defines only a finite part of the series there will be end effects in the series derived from it. These could be eliminated by topping and tailing, but as this usually serves only to obscure what is important, please just ignore these.

```
   f=.(,+/@(_2&{.))^:12(0 1)
0 1 1 2 3 5 8 13 21 34 55 89 144 233
   f^2             NB. Fibonacci squares
0 1 1 4 9 25 64 169 441 1156 3025 7921 20736 54289
```

Incrementing the cumulative sums and differences still leaves us in Fibonacci-land :

```
   >:+/\f                  NB. same as 2|.f
1 2 3 5 8 13 21 34 55 89 144 233 377 610
   >:-/\f                  NB. _2|.f, has alternating signs
1 0 1 _1 2 _3 5 _8 13 _21 34 _55 89 _144 0,
```

Multiplying terms of f which are neighbours but one :

```
   (*2&|.)f        NB. u(n) times u(n+2)
0 2 3 10 24 65 168 442 1155 3026 7920 20737 0 233
```

gives a series which differs by 1 in alternating directions from f^2.

Here is a selector verb which takes a binary pattern as left argument and extends it as a mask for the right argument. Its first use is to select odd and even items in f :

```
     sel=.($~#)#]
     ]od=.0 1 sel f             NB. odd items in f
  1 2 5 13 34 89 233
     ]ev=.1 0 sel f                  NB. even items in f
  0 1 3 8 21 55 144
```

Cumulating either odds or evens leads to the other :

```
     +/\od                      NB. same as 1|.ev
  1 3 8 21 55 144 377
     >:+/\ev                    NB. same as 1|.od
  1 2 5 13 34 89 233
```

and the Fibonacci trade mark is still present in the following :

```
     2+/\f                      NB. result is 2|.f
  1 2 3 5 8 13 21 34 55 89 144 233 377
     2-~/\f
  1 0 1 1 2 3 5 8 13 21 34 55 89
     _2+/\f                     NB. same as }.ev
  1 3 8 21 55 144 377
```

You can even generate the Fibonacci series using long division :

```
     (9!:11)16
     %9899
  0.0001010203050813214
```

Then try %998999 and do the long division (it isn't hard!) – you'll soon see how it works.

The products of pairs of consecutive items in the Fibonacci series generate the series :

```
     ]ppf=.2*/\ f
  0 1 2 6 15 40 104 273 714 1870 4895 12816 33552
```

which is the same as :

```
     +/\*:f                     NB. cumulate the squares of f
  0 1 2 6 15 40 104 273 714 1870 4895 12816 33552
```

Cumulating sums of adjacent product pairs of ppf brings us back to f :

```
    2+/\ppf                 NB. same as 1 |.ev
 1 3 8 21 55 144 377 987 2584 6765 17711 46368
```

Subtract successive products in pairs to get the odds, which are also the squares of f :

```
    2--/\ppf                NB. cf.0 1 sel+/\ppf
 1 1 4 9 25 64 169 441 1156 3025 7921 20736
```

The cumulative series of ppf is :

```
    ]cppf=.+/\ppf           NB. cum product pairs
 0 1 3 9 24 64 168 441 1155 3025 7920 20736 54288
```

odd and even subsets of whose items turn up in:

```
    ev^2                    NB. squares of even values
 0 1 9 64 441 3025 20736
    1 0 sel(*2&|.)f                 NB. prods of pairs one apart in f
 0 3 24 168 1155 7920 0
```

What happens when three or more successive items in f are cumulated ? :

```
    -:3+/\f                         NB. 2|.f
 1 2 3 5 8 13 21 34 55 89 144 233
    4+/\f
 4 7 11 18 29 47 76 123 199 322 521     NB. 2|.Lucas numbers
```

(The Lucas series is what the Fibonnaci series becomes if you start with 1 3:

```
    ]Lucas=.(,+/@(_2&{.))^:12(1 3)      NB. Lucas numbers
 1 3 4 7 11 18 29 47 76 123 199 322 521 843            )
```

Now replace + with - :

```
    |>:-/\f                         NB. _1|f
 1 0 1 1 2 3 5 8 13 21 34 55 89 144
    |2-/\f                                  NB. also _1|.f
 1 0 1 1 2 3 5 8 13 21 34 55 89
    -:3-/\f                         NB. f
 0 1 1 2 3 5 8 13 21 34 55 89
    |4-/\f                                  NB. _1|.Lucas
 2 1 3 4 7 11 18 29 47 76 123
```

Selecting and cumulating every third term leads to :

```
    +/\0 0 1 sel f       NB. cum sum of 3rd 6th, 9th items
 1 6 27 116
```

which should be compared with

```
    -:<:0 1 0 sel f           NB. half of f(3n+2)
 0 1 6 27 116
```

Fibonacci numbers have an intimate relationship with binomial coefficients as diagonal addition (`+//.`) on the Pascal triangle demonstrates :

```
    +//.(!/~)i.10
 1 1 2 3 5 8 13 21 34 55 88 133 176 189 155 92 37 9 1
```

If Pythagoras happens to be your t(r)ipple, try

```
    p=.1}.2*ppf             NB. 2 f(n+1).f(n+2)
    q=.f*3|.f          NB. f(n).f(n+3)
    r=.1}.2+/\f^2      NB. (f(n+1))^2 +(f(n+2))^2
    p,q,:r                  NB. Every col. is a Pythgn. triple
 2 4 12 30 80 208 546 1428 3740  9790 25632 67104   0   0
 0 3  5 16 39 105 272  715 1869  4896 12815     0 144 233
 2 5 13 34 89 233 610 1597 4181 10946 28657 75025   0   0
```

Fibonacci numbers have a closed form which uses the solution of the quadratic equation $1 + x - x^2 = 0$:

```
    ]gs=.(1&{::@p.)1 1 _1          NB. gs = golden section
 1.61803 _0.618034
```

An expression known as Binet's formula gives the nth. Fibonacci number as $\frac{1}{\sqrt{5}}(\phi^n - \varphi^n)$ where ϕ and φ are the items of `gs`, and so the Fibonacci numbers are given formulaically as :

```
    ]b=.1 _1*%%:5
 0.447214 _0.447214
    +/"1 b *"1(<gs)^&> i.15
 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

or equivalently, since the larger root (`gm=.-:>:%:5`) in `gs` rapidly dominates :

```
   rnd=.<.@(0.5&+)                   NB. round to nearest integer
   rnd (gm^i.15)%%:5
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

A closed form also allows the generation of large Fibonacci numbers :

```
   b+/ .*gs^78
8944394323791464
```

for which Roger Hui demonstrated another method (although not a closed form), viz.

```
   (9!:11)16
   {. |.@(+/\)^:(78) 0 1    NB. the 78th Fibonacci number
8944394323791464
```

The selection verb can also be used to demonstrate various divisibility properties of `f`, for example that even Fibonacci numbers have indices divisible by 3, all the others are odd :

```
   1 0 0 sel f
0 2 8 34 144
   0 1 1 sel f
1 1 3 5 13 21 55 89 233
```

This can be extended by generalizing the selection verb to provide complementary selections, e.g. to provide masks for integers in `i.4` which are/are not divisible by 4 :

```
   dpsel=.(0&=;0&~:)@:i.
   dpsel 4
+-------+-------+
|1 0 0 1|0 1 1 1|
+-------+-------+
```

In the next three cases, post-edited comments show the properties of the numbers in the various pairs of boxes :

```
    (dpsel 4)sel&.><f
+----------+-------------------------+
|0 3 21 144|1 1 2 5 8 13 34 55 89 233|NB.div/not div by 3
+----------+-------------------------+
   (dpsel 5)sel&.><f
+------+----------------------------+
|0 5 55|1 1 2 3 8 13 21 34 89 144 233|NB.div/not div by 5
+------+----------------------------+
   (dpsel 6)sel&.><f
+-------+---------------------------+
|0 8 144|1 1 2 3 5 13 21 34 55 89 233|NB.div/not div by 8
+-------+---------------------------+
```

All Fibonacci numbers which are divisible by 17 are even which is corroborated by :

```
   F=.(,+/@(_2&{.))^:60(0 1)
   17((=&0@:|)#])F
0 34 2584 196418 14930352 1134903170 86267571272
```

Continued fractions starting with 1, 1+1/1 are the result of cumulating the verb `+%` :

```
   (+%)/\13$1
1 2 1.5 1.66667 1.6 1.625 1.61538 1.61905 1.61765 1.61818 1.617
98 1.61806 1.61803
```

and converge rapidly to the golden mean. Applying rational arithmetic using `x:` gives :

```
   (+%)/\x: 13$1
1 2 3r2 5r3 8r5 13r8 21r13 34r21 55r34 89r55 144r89 233r144 377
r233
```

Multiplying by `f` keeps us marching on the Fibonacci spot (well, more or less!) :

```
   f*0,(+%)/\x: 13$1
0 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This also gives a convenient way to obtain some rather large Fibonacci numbers. The denominator has been edited out of the following display in order to show the 250th. Fibonacci number :

```
   (+%)/x:248$1
4880197746793002076754294951020699004973287771475874
```

No wonder there are a lot of rabbits about!

REVIEWS SECTION

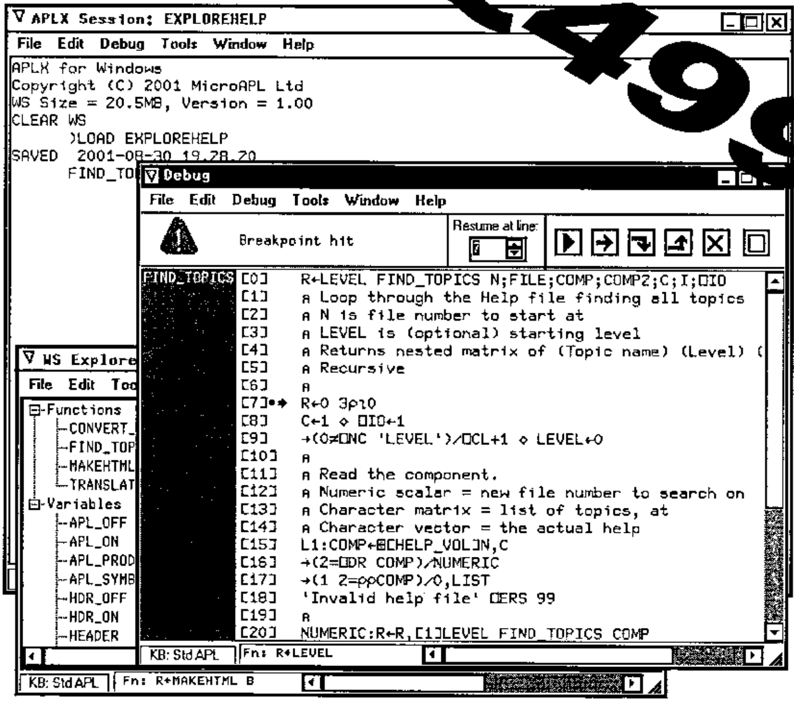
This section of Vector covers recent releases of APL interpreters, or software tools of general interest to the APL developer. If you have a new release of your own software to be reviewed, or you just find something which you think may be of interest, please contact the editor.

APLX

MICRO APL

for Mac OS 9 Windows 95/98/ME
Mac OS X Windows NT/2000/XP

£499



```
APLX for Windows
Copyright (C) 2001 MicroAPL Ltd
WS Size = 20.5MB, Version = 1.00
CLEAR WS
      )LOAD EXPLOREHELP
SAVED  2001-08-30 19.28.20
      FIND_TO
```

```
Breakpoint hit

FIND_TOPICS [0]    R←LEVEL FIND_TOPICS N;FILE;COMP;COMP2;C;I;⎕IO
            [1]    ⍝ Loop through the Help file finding all topics
            [2]    ⍝ N is file number to start at
            [3]    ⍝ LEVEL is (optional) starting level
            [4]    ⍝ Returns nested matrix of (Topic name) (Level) (
            [5]    ⍝ Recursive
            [6]    ⍝
            [7]•➔  R←0 3⍴⍳0
            [8]    C←1 ⋄ ⎕IO←1
            [9]    →(0≠⎕NC 'LEVEL')/⎕CL+1 ⋄ LEVEL←0
            [10]   ⍝
            [11]   ⍝ Read the component.
            [12]   ⍝ Numeric scalar = new file number to search on
            [13]   ⍝ Character matrix = list of topics, at
            [14]   ⍝ Character vector = the actual help
            [15]   L1:COMP←⍞[HELP_VOL]N,C
            [16]   →(2=⎕DR COMP)/NUMERIC
            [17]   →(1 2=⍴⍴COMP)/0,LIST
            [18]   'Invalid help file' ⎕ERS 99
            [19]   ⍝
            [20]   NUMERIC:R←R,[1]LEVEL FIND_TOPICS COMP
```

```
⊟-Functions
  ├-CONVERT_
  ├-FIND_TOP
  ├-MAKEHTML
  └-TRANSLAT
⊟-Variables
  ├-APL_OFF
  ├-APL_ON
  ├-APL_PROD
  ├-APL_SYMB
  ├-HDR_OFF
  ├-HDR_ON
  ├-HEADER
```

The new, cross-platform, full-featured, user-friendly, GUI-enabled, cost-effective APL.
Check it out at http://www.microapl.co.uk/apl

APLX and MicroAPL are trademarks of MicroAPL Ltd. Mac is a trademark of Apple Computer Inc. Windows, Windows NT ME & XP are trademarks of Microsoft Inc. Price shown excludes VAT.

APLX, the Awakening of the Sleeping Beauty

reviewed by Eke van Batenburg

Abstract

MicroAPL used to distribute APL68000, an APL system for the Macintosh. This year they have introduced APLX. This is a new APL that is available for Macintosh and for Windows. The workspaces from one platform can be read on another and (with a few exceptions) will also execute on the other machine.

This paper reports my experiences with both APLX versions.

History

In the beginning of the personal computer, screens had the same number of columns as a punch card and 640Kb memory was a ridiculously large amount of memory. It was a time when APL systems struggled with ROM chips to display the character set and a window was something that you looked through rather than at.

At that time the Atari and the Macintosh already had a very sophisticated APL: APL68000 distributed by MicroAPL. You could make nice dialog-boxes and windows with menus when users of DOS APL could only dream of such a user interface. Furthermore a programmer in APL68000 could use 1, 2 or even ridiculous amounts of 4 or 8 Mb memory whereas DOS users had to struggle with expanded or extended memory just to rise above the 640Mb barrier.

And although there were more than three APL vendors for the PC at that time and only one for Macintosh APL, I never felt the need for other vendors for Macintosh APL because this APL was a very good product.

In the following years the Windows operating system came to the PC and the Windows APL user interfacing grew more sophisticated year by year, using the possibilities that the development of the Windows O.S. offered. At the same time APL68000 did not develop very much which was OK because it was already far ahead of the other APL systems. In the last eight years however, development of APL68000 has come to a complete standstill, while in the meantime the Mac operating system moved from system 6, to 7, to 8 and to 9. The old APL68000 has kept on working, but now definitely lags behind in its functionality as it did not keep up with the developments in the Windows O.S. and the Mac O.S..

Recently Macintosh moved from O.S.9 to OS-X with a Unix kernel underneath which would probably make APL68000 eventually obsolete. So it was a pleasant surprise when I learned that MicroAPL suddenly surfaced with a completely reworked APL which they call APLX and which will work on the new Mac O.S.-X. It is my pleasure to report here how the first version of this new APL works.

Basic facts

APLX is developed by MicroAPL. You can find detailed information on their website //www.microapl.co.uk/apl.

There is a Macintosh version and a Windows version. Each one costs US$699 (£499+VAT). Previous APL68000 users get a discount for the Mac version (to US$575). Educational institutions get a 20% discount and for US$1400 they are allowed to distribute it among their students for free.

The Macintosh version runs under Mac O.S. 8.6 and upwards (9.2, OSX). The Windows version runs under W95, W98, ME, NT, 2000, 2000 Pro and XP.

A runtime version is currently missing, but will be available by the time you read this.

First Acquaintance

When you start APLX you see 6 menu items. There are no icons on the menu bar. These menu items are:

File. Here are [New], [Open], [Save], [SaveAs] and [COPY] to do the familiar system commands)CLEAR,)LOAD,)SAVE and)COPY interactively using the file entry dialog. Personally I would prefer to have the PCOPY rather than the COPY (or even better, both) but, of course, I can also type this command myself. This menu item also has the items [Import] and [Export] for the)IN and)OUT command, a [sessionwindow] item where you can save or retrieve the text and thus the history of the session window, a [PageSetup] and [Print] where you can choose between a print of the session window (or selected parts thereof) or a listing of all functions in the whole workspace. The latter item I found great to have available only a mouse click away.

Edit. The next menu is the Edit menu which contains the not so exciting, but useful options [Undo], [Cut], [Copy], [Paste], [Delete] and [Select All].

Here is also [Find/Change], [Find Again], a very helpful [Find Selected Text] and [Replace+FindNext]. Finally here is [Edit Recent] with recently edited items and [Edit Function/Operator/Variable] that will pop up a window where you can point and click the items that you want to edit.

Debug. The Debug menu has the options [Interrupt], [Resume Execution], [Clear Execution Stack] similar to [Ctrl+Break], →⎕LC and)SIC. There are also options [Show Debug Window], [Breakpoints] and [Watch] which I will talk about later.

Tool. The Tools menu has the options [Workspace Explorer], [Font Size], [Switch to non-APL Keyboard] to choose between classic or unified keyboard, [APL Keyboard Layout] which sets the keyboard layout to the classical or unified keyboard and [Preferences] which allows you to set defaults for many things from font size, colours, libraries, keyboard and so on.

Window. The Window menu has the options [Close Window] which in the session window is another way to exit APL, [Close Debug/Edit] to close other windows, [Tile], [Cascade], [Next Window], and a list of open windows.

Help., The Help menu has options [Contents], [Search for Help On], [Help On Selected Item], [Keyboard Layout] which gives a window with the keyboard layout (unfortunately one cannot click on a key to add it to the text in the session window) and a [Homepage]-activation and an [About] option.

The [Preferences] you find under the [APLX] menu (Mac) or [Tools] menu (Windows); the [About] is under the [APLX] menu (Mac) or [Help] menu (Windows) and the [Quit] (where would we be without this extremely important option) is on the Mac also under the [APLX] menu and in Windows (as Exit) at the bottom of the [File] menu.

Most options are self-evident, some of which I will elaborate on later.

Basics

Now first of all the basics: APL. There is not much to say about this because everything works just fine. APLX has the whole APL set of primitive functions with nested arrays. This includes the index (squeezed quad) and the find (underscored epsilon), first, n-wise reduction and scalar functions with axis.

Complex numbers are not implemented, nor are namespaces.

You can make your own user-defined operators.

For administrative applications there is available a very powerful formatting function "α", the standard format "⍕" can have a "picture"-type left argument and to top it off there is an extremely powerful function ⎕*FMT*. As I am not specialized in this area I have no personal experience with these functions, but the features in the description look impressive.

Another very powerful feature that APL68000 used to have is that with execute you could execute anything. Not only APL expressions, but also administrative commands. For example you could change the name of the workspace under program control by the expression:

```
    ⍎')WSID 1 ANOTHER'
```

I was happy to discover that APLX has retained this feature. The result is written to screen though, you can not say:

```
    RES←⍎')WSID'
```

All familiar *quad variables* and *quad functions* are available. There is not such a plethora as in APL2000 but most of those that are in IBM APL2 are available and more. A rough count taught me that APLX has about 90 quad variables/functions whereas APL2000 has 160 and 31 for IBM APL2 and 53 for APL2C. My count can be slightly off because my manuals are not all up to date and I was lazy and just counted everything (for example in APL2000 I chose the easy way and counted the silly duplication of quad ⎕*N*-functions and ⎕*XN*-functions as different functions), but the picture is clear. This is not to say that "the more quad-functions the better" (as I said, some quad-functions such as the ⎕*XN*-functions are plainly stupid) but it shows that APLX tries to give the user a good toolbox.

For error trapping there is no "⎕*ELX*", but there is a "⎕*EA*" and even more powerful alternatives such as error control "⎕*EC*" which executes the right argument and returns a nested vector with the result (which can be an error message) together with 1 or 0 (execution was fine or yielded error). There are other error trappings as well such as "→⎕*ERX #*" which sends control to line # in case of error.

All familiar *system* commands are available.

You can import and export workspaces using)IN and)OUT. I occasionally found it frustrating that in APL2000 I could not import one particular item. In APLX you can import the whole workspace, but you can also specify one or more objects that you are interested in.

For workspace I/O there are the familiar *)SAVE*, *)LOAD*, *)COPY*, but there are also the silent variants (starting with an extra S) that do not display a message and a *)XLOAD* that does not execute the latent expression.

Unfortunately there is no *)DISPLAY* nor *⎕DISPLAY* nor *]DISPLAY*. There is a function *DISPLAY* in a supplied workspace but I am not very happy with it. I often forget whether it is present in my active workspace or not. This results in a VALUE ERROR and subsequently in a frustrated feverish typing of the copy-command to get the *DISPLAY* available which interrupted my line of thought unfavorably. I hope that APLX will implement the)DISPLAY command soon.

In good old tradition, the path to a workspace is indicated by a library number. Associations between each number and its path can be set in the [Preferences] option. Another way is to specify the path using the monadic system function *⎕MOUNT*. You give it a character matrix where the first row refers to library 0, the second one to library 1 and so on. Unfortunately one cannot type the full path directly in the system command. This is a feature that I use very often in APL2000 for quick use to a not so common path, so I miss it dearly in APLX. Fortunately it will be available by the time you read this.

One of my long-standing wishes for APL2000 and APL2C is grouping. APLX has it all. For one, it has the classical grouping that uses the commands)GROUP,)GRP and)GRPS. It also has the modern grouping as available in IBM APL2 which is called indirect copying where you put parenthesis around a variable in the)COPY command to indicate that not the variable, but the names that the variable specifies should be copied. I often asked APL2000 to add this indirect copy to their system (or even the classical one), but up to now to no avail. So I am happy to find both in APLX.

Another grouping feature available is *⎕OV* which let you assemble a number of items (functions, operators, variables) in one named object. You can copy, save and retrieve this object and activate the items again.

You can make *user functions* and *user operators* in APLX.

There are many ways to invoke the editor.

One is to type [Apple+E] ([Ctrl+E] in Windows). This raises a window with a list of functions and variables. You can click on the particular function and it will open in the editor.

Another way is to activate the Workspace Explorer in the [Tools] menu. This opens a window with all your functions and variables at the left and its content showing on the right. Now you can immediately edit one or more functions. Unfortunately you cannot edit variables; the Workspace Explorer only shows you the "property" of a variable. For editing you need to leave the explorer.

The third way is the one that I use most often. Click on the name of the object (function or variable) in the session window and a popup menu opens that has Edit as its default action. Just release your mouse and you are in business for editing. Very smooth.

In APLX for Windows you have to *right*-click on the name. In APL2C and in APL2000 (after setting in the preferences) I can simply double click on a name to invoke the editor. I found the right click of APLX in the beginning only a small nuisance, but after some time it was very tiresome and I grew homesick to the Macintosh way or to the double click in APL2C and APL2000.

You can edit functions, operators and flat character arrays. You cannot edit numerical variables (alas), nested arrays or rank>2 arrays.

The *GUI* in APLX is different from its predecessor APL68000. Although a limited version of ⎕*WI* was already in APL68000, now all GUI is done by an extensive set of ⎕*WI* features. Most object names and properties in APLX are identical to those in APL2000 which makes conversion easy. The current collection of objects and properties is a solid base and good for most of the things a programmer might want to do. However many of the more luxurious things that a programmer might wish for are not available.

The graphics I consider the weakest part currently. The current number of objects and their properties that are now available is simply inadequate for serious work. For example one cannot place text at an angle along a vertical axis of a graph, nor can one draw (and fill) polylines. The new release 1.1 promises to remedy this weakness.

Experience

About APL itself I can be short. It runs fine. The APL system is stable and during all my testing it only crashed in two well defined cases (this is fixed in version 1.1). I also found problems with error trapping: ⎕*EA* does not always executes the alternative in case of error and ⎕*EC* does end prematurely in case of error. Apart from these few problems I found the system very reliable, a thing that I value highly.

Another thing is the GUI features. As I mentioned before, the most important objects are available, like radio-buttons, checkboxes, progress bars, exit buttons, trees and so on. A good thing is that APLX intends to be compatible with APL2000. For that reason there are radio-buttons and options, the same thing but with different names. My experience was that here are still some growing pains; things that work well normally but give trouble in edge conditions or in seldom used options. The good thing is however, that most of the problems that I found will be a thing of the past in version 1.1 which will be available at the time that you read this.

Out of curiosity I compared some APL functions in APLX with other APL systems. As I have no other APL systems on the Macintosh I did my benchmarking on a PC. I used APLX/Win on one and the same machine as the other APL alternatives to make the results comparable. You can see the result:

	APLX/W	APL2000/W	APL2C	IBMAPL2
⍞←7	0	6.60	10	3.3
⎕←70ρ'⌹⍫⍒⍋⌽⍉⊖'	2.2	8.20	10	10.5
Q←+/VI	0.5	0.60	0	0.5
Q←+/VR	0.5	0.50	0	0
Q←∨/VL	1.1	0.00	0	2.2
Q←⌈/VI	0.6	0.60	10	0
Q←⌈/VR	0.5	0.50	0	1.1
Q←VI⍳MI	17.1	6.10	490	8.2
Q←VR⍳VR	6555.3	18.10	1510	3197.8
Q←VR=⌽VR	7.7	1.60	0	3.3
Q←?10000ρ1000	44	9.40	20	13.7
Q←10000?10000	45	6.60	20	13.8
Q←VI*0.5	4.9	9.30	10	7.7
Q←7∣VR	114.3	11.50	0	6.5

Q←VR[IVR]	3.3	1.10	0	11.6
Q←⍒VI	17.6	7.70	0	2.8
Q←⍋VR	19.2	2.20	10	9.9
Q←¯20 0↑MR	0.5	0.00	0	0
Q←VI∊VI	9.9	3.90	220	4.9
Q←(10000⍴1)∊1000⍴ 0	8.8	2.70	270	0.6
Q←MC∊MC	7.1	1.70	10	2.7
Q←21⍉MC	9.9	1.60	0	0.5
Q←VC∘.=VC	11	2.20	80	2.2
Q←(⍳300)∘.+⍳300	14.3	20.90	80	11.5
Q←VR⌊.+VR	23.6	2.20	10	3.9
Q←MR⌹100↑VR	84.6	3.30	0	10.4
Q←tCPUBENCH_FIBON	80.2	3.80	30	11.5
-LowerQ.	0.85	0.85	0	0.85
-Median	9.35	2.45	10	3.6
-UpperQ.	21.4	7.15	25	10.45
-Mean	262.36	4.92	103.33	123.74
-GeometricMean	9.29	3.19	39.4	5.47

If you look at these figures or at the summarising median (or geometric mean, as the normal mean is not very helpful with skewed distributions or with outliers) you see that in general APLX is reasonable in its speed. Some functions are a bit slow, some are not. One very distinctive feature is that output is extremely fast in APLX. The other APLs are roughly similar in speed, but APLX beats them hands down. This agrees with the impression that APLX gives, it seems to react very fast, probably due to its fast output.

Another remarkable thing is dyadic iota. You see that APL2000 is lightningly fast (the number 18 is really no mistake) and on the other hand APLX is extremely slow. I am really surprised how APL2000 managed to be so extremely fast and APLX so very slow. These figures are comparable as the benchmarks are on one and the same machine within a few minutes of each other.

One other observation is that APLX seems very dependent on machine power. On a small first generation Pentium machine of 133MHz with only 32Mb memory under Windows98 various development tasks (such as showing the edit-window or debugging window) and file I/O took a *very* long time to accomplish whereas APL2C and APL2000 still performed well. On more powerful machines this difference evaporated.

MicroAPL is working on this and expects various speed-ups in version 1.1 and subsequent releases.

Documentation

For problems you have the following help.

First of all, there is the previously mentioned [Help] menu.

The [Help] is very useful. In APL2000 I always have to determine beforehand if the item I want help for is a function, a windows type of thing, or a session-type of thing. Here I can just activate [SearchOn] and type the item that I need help for. Unfortunately the [SearchOnSelected] is rather restricted. If you type ⍴ followed by [F1] nothing happens; you need (as the menu option prescribes) to actually highlight the character to activate the [SearchOnSelected]. Also words like "style" or "Form" or ")SAVE" will not rouse the [SearchOnSelected] from its sleep. For this you have to activate the [SearchOnHelpFor] and type in the topic that interests you. So, although the principle of [SearchOnSelect] has promise, it needs more work to make it really useful.

The [SearchOn]-Help works well on the PC, but not on the Mac. On the Mac you cannot type APL characters which makes it worthless for APL help. Moreover the help does not react well to GUI-related keywords; for example asking help for "scale" gives me help from QuickTime but not for APL, asking help for "Form" gives you AppleScript help and typing)SAVE warns you that the search phrase is invalid. Here some work is needed in the Mac version whereas the PC version works well.

Your second help is *documentation*. There are two manuals provided in PDF format. The first one is the "APL language manual" [1]. This is very thorough and useful. Although it is not written as a tutor but rather as a reference, I think that a beginner can learn APL quite well, in particular because of the wealth of small examples. But the experienced programmer will also find many paragraphs that are useful to study such as details on how to do error trapping, or on working with component files.

The other manual is the "APL GUI programming manual" [2]. This describes all objects, methods, properties and callbacks. This is essential if you want to build a GUI interface. Many paragraphs are fine, with a good description and useful examples, but some paragraphs are slightly too condensed to my taste (for example the page with "style" description is nearly empty and refers the reader to the objects, but on one occasion I found a style value that worked but was not described in the object paragraph). I trust that the next version will be a bit more extensive.

One very useful help that I consulted often are two supplied *demonstration* workspaces. The first one is HELPQWI which gives small examples of each widget (*⎕WI* class) that is defined. The second one is SAMPLESQWI that gives a few more extensive examples. Most of these examples were pieces of APL code from the manuals, but I found the functions in these workspaces very enlightening and useful to modify.

Compatibility

Compatibility with older version

First of all I was interested whether my old workspaces would still run in Mac O.S.-X or not. It appeared that I could load my old workspaces without any problems. In all APL programs the pure APL expressions worked flawlessly.

A very tricky thing was the user interface. APL68000 had many (machine-coded?) functions for nice GUI interfacing and these were used in many of my old programs. Would they still work? No, not immediately, but APLX provides workspaces with the replacement functions.

I replaced the old utilities with their new O.S.X counterparts and...... magic: everything worked fine.

Compatibility with other APL systems

Another action I took was to port some of my APL2000 workspaces to APLX/Mac. I used the]OUT in APL2000 and)IN in APLX/Mac. Most functions ported fine, some did not. I found that functions with ⎕ELX in the header were not imported and (more seriously) were not reported as being problematic. After I found out about this problem I removed the culprit in the headers before exporting and converting then went well. I hope APLX will address this problem and store "unfixable" programs as ∆∆∆-variables in the workspace together with an appropriate message.

Of course, when I say that everything went well I meant that the pure APL code worked fine. I noticed to my surprise how often I used the double quote character in a string instead of two quote characters, which is not defined in APLX. Such incompatibilities have to be rectified before you can run your APL2000 code in APLX, of course. Or not, as MicroAPL promises to have this also implemented in their next release 1.1

Another point is the ⎕WI code. I was happy to discover that APLX introduced ⎕WI for all the GUI interfacing. In many ways this is compatible with APL2000. In some ways they are not similar because the implementers of APLX chose a better implementation. For example to communicate with the clipboard, APL2000 has several lines of ugly non-⎕WI code; whereas APLX applies a very nice:

```
    '⎕'⎕WI'text' 'TO CLIPBOARD' ⍝ write to clipboard
    '⎕'⎕WI'text' ⍝ read from clipboard
```

On the other hand, APL2000 has developed over the years quite a collection of ⎕WI features; not all of them have made their way into APLX yet, but I suspect in the coming months APLX will quickly come up to par.

Compatibility with APLX /Win

One claim that I was rather skeptical of is that APLX can read/write their workspaces irrespective of Mac or PC. I saved a PC workspace, and (after removing the dot in the name) loaded it from floppy into the MAC and surprisingly it worked!! This is very neat. Later on I learned that it works only for workspaces that are saved with clear status indicator so you should be aware of this when you save a workspace for use on the other platform.

There are a few exceptions. For example there is no Rich Edit object, no Choose Color dialog nor popup menus for the Macintosh.

Development tools

One of the things that I became very fond of in APL2000 is the debugging window. I was very happy to see that APLX also has this tool added to its APL. One thing that I like very much in APLX is that you can edit the code in the debug window. This is still (apparently I am a slow learner) a trap for me in APL2000: when my program crashes and the debugging window shows the problem code, my immediate response is to click in the offensive line in the debugging window to make corrections. Then I realize that this does not work because I have to activate the edit-window instead. In APLX however, my gut reaction works just fine (another reason why I will probably never learn that it does not work that way in APL2000).

This debugging window works much more pleasantly on the Mac than on the PC. On the PC, setting and removing stops requires a double click (contrary to the Mac where one click is enough). I fail to see why you need a double click in APLX/Win, whereas in APLX/Mac and APL2000/Win a single click works just fine. Similar to the right-click that APL/X requires for editing instead of a double click, this seems a small thing (and initially it *is* a small thing) but as you do this often during development it becomes tiresome in the long run.

Also there is something not quite right in the windowing system that APLX/Win uses on the PC. If you have a debugging window, or an edit window, or a help window, you cannot easily go back to the session window (something that you frequently need to do) by clicking on its reference on the Start-bar underneath. This reference in the Start-bar is insensitive to clicking. So in order to switch back to the session window you have to use [Alt+Tab] or activate the [Window] menu. The reverse is no problem, going from the session window to another (edit- or debug-) window by clicking on its reference in the Start-bar works well, but you cannot go back in the same way. In the new release one has a quick way to switch windows using [Ctrl+#] where # is the window number (0 for session window), but this does not work when being active in the help-window or in another application.

There is also a Watch facility, which is basic, but useful. You specify the variable that you are interested in and the watch window continues to show its latest value (which can be a VALUE ERROR as long as the variable is not yet used). There are no fancy extras here (like conditional expressions or specifying the scope) as there are in APL2000 but for me this worked fine.

Unique Assets

One of the things that I already admired in APL68000 is the component file system. It is still available in APLX. The beauty is that it is very APL-like with only four symbols that do all you need: quad-write ⍈ for writing, quad-read ⍇ for reading (data or file information), quad-drop ⍗ for deleting (components of) a file and quad-hold ⍐ for setting quota and for setting sharing rights.

Another thing that looks great, although I do not have any experience in it yet, is the possibility to activate external functions. In APL2000 I had several tries to communicate with DLLs with ⎕*NA*, but I always had to give up because of spurious unexplainable data-transfer hiccoughs (sometimes perfect data transfer, the next day only half the array transferred). I certainly will try and see if APLX will do better.

A good thing is that APLX does not gratuitously copy other GUI solutions. I mentioned the clipboard solution. They also have an easy Choose Color object, an object to choose fonts and an object to activate the file-dialog box. They also have a nice feature called "anchors". Normally the "where"-property fixes the distance of an object to the left- and upper side of its form regardless of any resizing of the form. With anchors you can alternatively fix the distance to other sides, for example to keep the exit buttons on a fixed distance from the lower edge of the form. Also they are considering a relative anchoring which would make most of the "on Resize" code obsolete.

I expect more of their future extensions to be implemented with the same well-thought approach.

Wishes

Do I have any wishes? You bet. Although such a rarity as a happy user might exist, I do not think that a satisfied user exists.

My most modest wish that APL2000 never wanted to grant me (but which is to my satisfaction in APL2C) is *lev* ⊣ and *dex* ⊢. Though extremely trivial at first sight I use them a lot in APL2C to suppress unneeded output and to perform several expressions in one line. The reason is that I like

```
    subexpression2 ⊣ subexpression1
```

much more than

```
    subexpression1 ⋄ subsexpression2
```

because the first expression reads from right to left whereas the latter reads from halfway right to left and subsequent from right to halfway left.

I also hope for the *duplicate*, the *commute* and the *rank* operator, and the *LCM* and *GCM* functions which are all in APL2C, but are missing in APL2000 and in APLX and for the *unique*.

All these functions are part of the extended ISO standard [3] so in my opinion APLX as well as APL2000 should implement them.

Another thing that I missed are user commands (APL2000 terminology) or external functions (APL2C terminology). I like the implementation of the latter most. I like it because of its simplicity; you just ask APL to write the vector representations of the functions to file, for example:

```
(⎕VR'MAIN')(⎕VR'F2')(⎕VR'F3')(⎕VR'F4')⎕IIO'C:\PATH\DoMe.EFN'
```

and now hencewith you can type:

```
]DoMe  xxx
```

This command will read the functions from the file `DoMe`, fix them and activate the first function (in our example `MAIN`) with argument `xxx` and delete them afterwards (in the workspace, not in the file, of course).

If I should be granted three wishes, my last one would be to add proper control structures. I hope that APLX will implement true APL-like control as it is implemented in APL2C rather then the horrible BASIC-like elaborate wording that is implemented in APL2000. In APL2C you can formulate:

Iteration

```
∇4+5
   ⎕←'HELLO'
   ⎕←'WORLD'
∇
```

This is a block that is executed 9 times:

Condition

Similarly for conditions, the right control argument is either one or zero. For example, to do something only if A is equal to zero:

```
∇0=A
   ⎕←'HELLO'
```

```
    ⎕←'WORLD'
  ∇
```

While

The while-construction is implemented in a similar way. After all, a while-construct is an iteration, but an iteration that is executed as long as the controlling expression is true. Therefore the implementation is identical to the fixed iteration however, to tell APL to recheck the controlling expression a minus one is used:

```
∇-0<S
  ⎕←'HELLO'
  ⎕←'WORLD'
∇
```

The inner block will be repeated as long as variable S is positive.

Alternatives

For alternatives like IF-ELSE and SELECT, control takes a vector for its right argument. This vector prescribes what to do for each alternative, which is marked by a label-less colon ":". An example illustrates this:

```
∇2 3=⎕NC'XXX'
:⎕←'XXX IS VARIABLE'
 ...(other code in case of variabe)...
:⎕←'XXX IS FUNCTION'
 ...(other code in case of function)...
∇
```

My experience with these control structures in APL2C has shown me that such code is read much more quickly than the BASIC-like wordings that are used in APL2000. Even more so for the single-line alternative which will be in APL2C shortly:

```
⎕←'VAR' : ⎕←'FUNC' ∇ 2 3=⎕NC'NAME'
```

rather than the elaborate:

```
:IF 2=⎕NC'NAME'
:THEN
  ⎕←'VAR'
:ELSE
  ⎕←'FUNC' ∇
:ENDIF
```

Why move to APLX?

If you are happy with your current APL then there is no need for you to move.

However if you wish to develop your application for two platforms (Mac and PC/Win) this is your opportunity. You can just restrict yourself to development on one platform until it is finished and then move the workspace to the other platform. Switching back and forth between the two platforms is a breeze. You can easily load any workspace from the other platform and if you programmed your graphical user interface with the `⎕WI` there is no need for you to do *any* conversion.

If you want to have your APL2000/Win application running on the Mac, APLX/Mac is also a good choice. It is likely that you will need to reprogram some code for the graphical user interface, the control structures and some new code for a few of the manifold quad-functions that are in APL2000. A good thing is that many of the `⎕WI` functions of APL2000 are there and many quad-functions of APL2000 are also in APLX, but not all. So to do this you need to invest some effort. However all ISO programmed code will run well without any problem.

If you are working with an old-fashioned APL system (DOS, APL-II) and are considering moving to a professional APL this is a good choice. Although not inexpensive for private users, its price is affordable compared to the other APLs. If you can export your old APL code to a transfer file, you can port your code into APLX using `)IN`. As you will not have used GUI-code nor control structures in your DOS APL, chances are high that you will not need to do much conversion. I also expect that IBMs APL2 will run with little or no changes, except for the auxiliary processors of course.

One of the things that I liked very much about the J team is that they are very responsive to user comments. That does not mean that they gratuitously accept any suggestion that users make. Of course they don't, but they are interested in listening to opinions and eager to improve and enhance their product. During the few months that I worked with APLX and reported my experiences, I found the same attitude with the microAPL team. This was already the case, long ago when they developed APL68000, so this gives me great confidence in the future of this product.

Conclusion

I have worked for a few months with both APLX/Mac and APLX/Win. There were no problems in porting my older Macintosh workspaces to APLX/Mac. And after copying the updated GUI-functions the resulting workspaces run fine.

To have my APL2000/Win code running on APLX/Win (and subsequently on Mac) took a bit more effort. Many of the GUI-coded lines did run immediately, but there were also quite a few differences, often missing features but sometimes better alternative implementations though.

Apart from this, and apart from a few well defined problems with error trapping and crashes when using progress bars, my experience with APLX was very good. It is stable, which I value highly. The APL functions, operators and variables are all present and work as they should. The available GUI features are powerful although not extensive and I hope that the coming year will bring them up to a higher level. In particular the graphics felt inadequate for me.

New development in APLX was pleasant. The development tools (debugging window and watch window) are very useful. Some rough edges have to be smoothed out in the coming months. The available documentation is good: the PDF documents are well written, the [Help] in Windows is good (the Mac version needs work) and the supplied DEMO workspaces were also very useful to look at and to modify or to copy some code from.

My biggest joy was that I could simply load a Windows workspace on the Mac and see it running without any reprogramming on my part. This was pure magic.

Literature

[1] MicroAPL (jan. 2002) APLX language manual (v.1.0). MicroAPL Ltd.

[2] MicroAPL (jan. 2002) APLX GUI programming manual (v.1.0). MicroAPL Ltd.

[3] ISO (1997): Programming languages, their environments and system software interfaces – Programming language APL, extended. ISO13751.

Further comments from Anthony Camacho

[I was working on a review of the PC version of APLX when Eke's comprehensive review of both PC and Mac versions arrived. I've been asked to add, from my notes, anything that Eke doesn't say.]

1 Setup ran through without a hitch in less than five minutes. I chose a different directory from standard and did a custom install and chose every option.

2 I'm one of those strange people that reads manuals (perhaps because I used to write them) so I began with the documentation (which is on .PDF files). After reading the introductory bit telling me how nice APLX is (it offers syntax colouring, pop-up objects ie function or variable editing, a "watch window" to show when a variable changes and a debug window and I also learned that its

workspaces are the same whichever the platform you run it on) I began on the manual before even loading the interpreter.

2 I was delighted to discover I would have the permanently available instant choice of unified, traditional or standard text keyboard.

3 The first three chapters in the APL Fundamentals section of the APL Language Reference are *Array type and prototype, Axis operator* and *Binding strengths.* I was amazed because so many previous attempts to introduce advanced APLs to me had done their best to avoid raising these awkward subjects as long as possible. I've never before read such a good introduction to an advanced APL. I was able to read it like a novel, the same way I read Gilman and Rose when I first heard about APL and was advised to learn it. I guess that revelation was sometime in 1979; on that occasion I put out the light after 3 am! I am reminded of Einstein's dictum "Everything should be made as simple as possible, but no simpler" (for which I cannot now find the reference). I have encountered many explanations of complex subjects which fail, for me, because they begin by over-simplifying, which makes them untrue, and so reading them again, when one knows better, shows them up. This one, spectacularly, neither over-simplifies nor skips or fudges the difficulty, yet remains readable. In short I think this introduction succeeds brilliantly. It is worthy to stand alongside Paul Berry's Sharp manual and, if anything, is better than Gilman and Rose.

4 I had not previously come across the *nomadic* function which I had always previously called *ambivalent.* I think nomadic is a poor name. Such a function does not wander.

5 Even a really devoted manual reader like me eventually has to try something, so I did start APLX, by clicking on APLX.EXE. Then the session window opened with the notice:

```
APLX for Windows
Copyright (C) 2001 MicroAPL Ltd
WS Size = 20.0MB, Version = 1.01
CLEAR WS
```

The first thing I wanted to do was to see if the APL was there and so I produced some tables and looked at quad-AV and so on. I wanted to look at the supplied workspaces and couldn't find them. Then I discovered quad-MOUNT. This provides the facility to assign a path to a library number for up to ten libraries. Then)LIB <n> will return the workspaces in that subdirectory. In the [Tools, Preferences] window one has the option to set the start-up (quad-MOUNT) paths of up to ten libraries. Very conveniently, if one uses the Preferences, the paths entered are immediately available and are set every time APLX is started.

My experience with the interpreter matches Eke's: there is simply nothing to say because everything I tried works perfectly. I'm a very old APLer and have never made significant use of anything more advanced than Sharp version 20, so I've not yet looked at defined operators nor have I done anything like an adequate test of nested arrays.

6 There are occasional lapses in the manual. For example I found, in the description of operators, some examples using A and B for right and left arguments just after a sentence saying L and R would be used. At the end of APL Fundamentals, when the "next" link isn't there, one has to choose between "previous" and "contents"; "contents", instead of taking one back to the contents of the APLX Language Reference or "Help on APL Language" as the screen is headed, takes one back to the APLX Help where the language reference is one of the choices:

  Using APLX for Windows
  The APLX keyboard
  APLX language reference
  APLX Objects
  Interfacing to other languages

  Strangely, after doing this for Fundamentals and Primitives when it comes to Errors, File System, Native File Functions, and System Commands there is a "next" option which steps on to the beginning of the following section painlessly.

  The MicroAPL team were very helpful and I believe none of the problems I found above will still occur in version 1.1. For that reason I will not bore you with any more of the minor things I found.

7 The heading of the help window while going through the APL primitives cannot display some of the APL characters. The first to fail is floor, but there are many. Iota is displayed as ¼ and the file functions as a capital O with various accents. I suspected that this is one of the limitations of Windows - that you cannot choose just any font for the windows header line and MicroAPL confirmed it. This cannot be corrected because it is a limitation of Windows.

In short this is an APL interpreter out of the top drawer. When it has the free run-time and other enhancements promised for version 1.1 it should go on anyone's short list (provided complex numbers are not essential): in *"Which?"* terminology it is a best buy.

Dyalog.NET – First Impressions

by Ray Cannon (ray_cannon @ compuserve.com)

Disclaimer

This is my view of ".NET" (pronounced "dot-net") as a practical APL programmer. Due to time restraints (Dyalog.NET becoming available and Vector being sent off to the printers, ready for distribution at the Madrid 2002 conference) this is by no means a "full review".

Some Questions

When I started out with Dyalog.NET, what I wanted to know was; What is it? What it good for? How easy is it to use? Is it portable? Will it be worth investing my time in learning? What's the learning curve? Will I enjoy using it? Will I be able to put up my charging rates as a ".NET" APL consultant?

Don't expect me to be able to answer these questions for you, its hard enough answering them for myself, but at least I can share my impressions with you.

What is this ".NET"

Microsoft's ".NET Platform" is a completely new API (Application Programming Interface). An API provides a programmer with "hooks" into the utilities and services supplied as part of the (extended) operating system/platform. ".NET" is I guess, an order of magnitude bigger (in terms of the number of utility calls available) than the 32bit Windows API. ".NET" is fully "Object-Oriented" whatever that means.

What Dyalog have done, is to make all of these API call available from within the Dyalog APL environment via a single new system function "USING" which only requires the name of the ".NET namespace" and the name of the file (a DLL) as arguments. The DLL name is not even required for some "core" utilities. So, once you have said what ".NET namespace" you want, (and if necessary where to find it), you can use all the "utilities" within that namespace from APL.

Not only is this new API much bigger, it is also constantly growing, as it can be added to by anyone writing in a ".NET" language. So you and I (as a Dyalog.NET programmer) can write new ".NET" utilities, Microsoft can write new utilities, and third parties can write new utilities AND WE CAN ALL USE THEM. All you need is the DLL file installed on your PC.

(".NET" is much more then just a set of utilities, it's a concept, a paradigm, an environment, a protocol, a discipline. But if we just stick to the idea that it is a very large set of new utilities, we have something tangible, and immediately useful.)

Prerequisites

"OK, you sold the idea of .NET to me, what do I need?"

(Lifted straight from Dyadic's Dyalog.Net manual)

> Dyalog.NET requires a computer running Windows 2000 or Windows XP Professional with the following installed:
>
> - Dyalog APL/W Version 9.0
> - The Microsoft .NET Framework SDK version V1.0.3705 or higher.
> - Microsoft Internet Information Services (IIS) 5.0 or 5.1
> - Microsoft Internet Explorer version 6.00.

Please note the Win2000/XP requirement. I do not know what the restrictions are caused by, but UNICODE is certainly one of them. However, I believe a resulting Dyalog.NET application can be run under Win98SE, if it has been upgraded with ".NET". It will never run under Win95, as Microsoft have stopped supporting that OS.

Installation

Note that the Dyalog.NET CD does not install any files which are unchanged from version 9.0 (such as "win.dot"), hence part of the requirement is having Dyalog 9.0 already installed.

Having just installed the whole of Microsoft's Visual Studio.NET and the Framework SDK (a massive 4Gb), the installation of Dyalog.Net was a doddle (that is quick and easy).

The only problem I had was when I installed the "Dyalog APL Input Method Editor". (This allows you to type APL characters in Notepad for use in "scripts". It uses the same standard techniques as those used to enter Japanese characters in Notepad.)

By error, I selected (via a Microsoft wizard) the option to make Dyalog APL my "default" input method. In this context, APL is, to coin a phrase, "the wrong type of Language".

On re-starting Windows 2000, the system hung, being unable to find the correct file ~~to~~ with which to start Windows up in "APL" (as against starting Windows up in English or Japanese).

This was NOT an error that Dyadic have any real control over, it's a problem with Windows 2000, and XP (as I found out when I told John Daintree about it, and he said that he has had the same problem under XP at home).

Getting stared – "RTFM"

It always a problem when you start learning a new language. You type in the instruction to get "Hello world" displayed, but what next? Well reading Dyalog's manual (and trying it out on the PC) is a good start. But where do you go for help?

I started by joining the Dyalog.NET mailing-list. It'~~d~~s friendly and very helpful.

Initially, I made some fundamentally wrong assumptions, which caused me to ask for the same information (which I already had) a couple of times. The members of the mailing list were helpful, and quickly pointed in the right direction.

What I wanted to do was use the SetPixel "method" of the Bitmap "class". Where can I find it?

When using Microsoft's documentation about the "methods" (functions) in a "class" (utility sub-system), you will find there is no information about the name of the DLL file (the file containing the "class" that the "method" is a member of) in the method documentation. You will however find this information is available in the documentation for the class.

The Bitmap class is in the file "SYSTEMS.DRAWING.DLL".

Another part of my problem was that I could not find this file, using the standard Windows "SEARCH" item on the popup menu from the "START" button. Searching, starting from my "C:" drive produced no results.

I eventually found it by starting my search from "C:\WINNT\Microsoft.NET".

C:\WINNT\Microsoft.NET\Framework\v.1.0.3705\SYSTEMS.DRAWING.DLL on my machine, contains the "Bitmap" class, that contains the "SetPixel" method I wanted.

My first project

Since "Hello world" hardly qualifies as a project, I decided to write a Dyalog.NET application that would generate Mandelbrot bitmaps. (Hence the "SetPixel".)

I wrote my very first Mandelbrot generator in the early 1980's in Basic on a BBC Computer (with a 6502 chip). It took four plus hours to create a very small four-colour image.

I wrote a DOS based version for 386 chip in C plus assembler which took about a minute. I then wrote a Windows (3.0) version which run in sub 1-second time on a 486 or better.

With a 1GHz Pentium, and using tail-recursive "dfns" under Dyalog, I thought I might be able to speed things up a bit from my Basic version.

It did, but, I quickly found that to create a decent image, even with tail recursion, APL was not fast enough. (I was initially "recursing" up to 10,000 times for each pixel in the "black hole" areas of the Mandelbrot image. Some images require and even higher number.)

So I wrote a small programme in C# to do the intensive calculation required to calculate the "value" for each pixel.

C#

Microsoft's new language "C#" has been developed as the "recommended" development language for "Microsoft.NET". Charles Petzold has even re-written his classic book for C#, which is now called "Programming Microsoft Windows with C#" - Microsoft Press, ISBN 0-7356-1370-2. (I still have my 1992 copy of "Petzold" written for Windows 3.0 and have this latest one on order.)

After trying my hand in the past at Borland C++, I found C# (under the Visual .NET IDE) quite straight forward. I had to create a new "class" and save it as a DLL file. After a couple of false starts, I got the hang of the IDE. I can now edit the source code C#, re-compile it into a DLL, and test it from APL in a minute or so.

Calling my "home-made" classes was no different from calling those supplied by Microsoft. First I had to say I wanted to "also" use my DLL:

```
    ⎕USING ,←⊂'Manb,c:\bin\debug\Manb.dll'
```

Then I had to initialise my class (called "Manbot")

```
  m←Manbot.New(,ymax)(,xmax)
```

This created a new namespace called "m", by running the "New" (constructor) method, with two arguments.

When it came to calling the main routine (ManColour), all I needed was

```
  colours←m.ManColour⊂ args
```

This ran the "ManColour" method, with the supplied arguments returning the "colours" I required.

I can now do simple 320x240x32bit colour Mandelbrot image in under a second from Dyalog.NET

The time cost of calling the C# for a single pixel, was greater than doing the calculation from within APL. However, calling C# once to calculate ALL the pixels, was much faster. But this is still slower than my C++ version with its hand crafted in-line floating-point assembler.

(C# does not support in-line assembler.)

Problems and Feedback.

As with any new software product, there will be bugs and problems. Dyalog.NET was no exception.

I had problems with creating a BITMAP from Dyalog.NET. The code worked fine from Dyalog APL/W 9.0.3 but failed to create a valid "bmp" file from "Dyalog.NET Release 1". This was due to an incorrect header in the "bmp" file.

I reported the problem back to Dyadic. They put a fix in the first update for this, which I have now installed, and it is now all working fine.

(In the mean time I starting using "PNG" rather than "BMP", with the result that the size of my image files went down from 906 KB to 66KB.)

Conclusions.

What is it? A set of utilities.

What it good for? Anything Dyalog APL can do plus a whole lot more.

How easy is it to use? For an APL programmer, compared to C#, much easier.

Is it portable? No - it will only run on ".NET" enabled systems.

Will it be worth investing my time in learning? Yes I think so.

What's the learning curve? With the right support, gentle enough to ride a bicycle up. I might sweat a little, but not so steep that I will have to "get off and push".

Will I enjoy using it? I hope so, I have so far.

Will I be able to put up my charging rates as a ".NET" APL consultant? I intend to, as soon as my client starts using a Microsoft.Net environment.

APL+Win Version 4.0

reviewed by Michael Hughes (Michael@Hughes.uk.com)

My upgrade copy of APL+Win 4.0 has now arrived and although I have been playing with the beta version for some time it is still exciting to receive the "real" thing.

The installation of the upgrade went very smoothly, automatically registering the new Grid OCX and APL+Win as a COM server. I had one minor problem after the installation when the APL font did not load properly when I started up the APL session. It was easily fixed by resaving the screen font information in the ini file using the "Options/Font for Screen..." menu. The actual APL+Win fonts had installed successfully, it was due to the screen font settings not being updated in the aplw.ini file.

The product appears stable and seems to perform well. To anyone new to APL+Win it is probably best described as a close implementation of APL2 with a VB style interface to the Windows GUI using the system function wi. There is a lower level C style interface using the system function wcall but it is normal to use the wi interface. Occasionally it is useful and sometimes necessary (becoming rarer with each new version) to use the lower level access provided by wcall. All the GUI objects include their window handles as a property to simplify the use of wcall. It is now possible to retrieve the APL+Win GUI name from its handle.

The main new features in 4.0 are;

Three volume paper documentation suite;

User Manual

System Function Manual

Windows Reference Guide

New APL grid OCX

Minor changes to the session manager

Enhancements to the standard GUI objects, including a context help facility

Overhaul of the Printer object

Small interpreter/system changes

The paper documentation comes in the slightly larger than A4 size, originally used for the Windows documentation for version 3.6. This is the first time all the manuals have come in printed form as standard since version 1.0. This reprinting has been long overdue as the interim versions have been confusing to new purchasers. I have had to lend out my version 1.0 manuals on several occasions.

The documentation is clear except that the Grid object has been omitted, although it is covered in the online help. This is a serious omission but I'm assured that a printable copy is to be made available on the APL2000 website (www.apl2000.com) as soon as it is ready.

The manuals are useful for someone new to APL, or as a quick reference to the more experienced, as they cover all aspects of the primitives, system functions and variables. I found the explanations they gave were clear and to the point. The only difficulty is sometimes deciding which manual to go to, as, in my opinion, the arrangement is not always the most logical. They contain basic descriptions but also contain other aspects of windows programming which again would prove useful for people new to Windows.

The Grid is a great addition to the system and has been introduced, I suspect, because of the upgrade problems with the Formula One Grid. The F1 grid had previously been the usual solution if a general purpose grid was required. The new Grid appears to be similar to the F1 grid but has several advantages over the F1 grid. These can be summarised as;

> Comes as part of the APL+Win product and so does not involve separate (and now expensive) licensing and the upgrade paths of the two should now remain in step.
>
> The ability to allow the editing of frozen cells, in F1 they were protected.
>
> The data cells are accessed in an APL friendly manner using arrays
>
> The setting of attributes are managed with APL arrays rather than by continually changing selections, applying the attributes and resetting the selection.

The grid also seems efficient and fits in very well with the rest of the APL system. It can generate and read data in XML format and is available for use by JAVASCRIPT as well. I have almost finished converting my F1 applications to the new grid and all seems to be fine.

I think the lack of paper documentation and the fact that the Print Preview method is still APL code rather than a method indicates that the grid development was not really finished in time for the release. I hope these deficits will be corrected quickly and that other enhancements like the ability to detect the mouse over the bitmaps in the corners of a cell will be added soon. These are, however, minor complaints in the face of a generally good piece of work from APL2000.

The system itself has been extended so that

> replicate now matches the APL2 definition where negative values in the left argument insert the modulo number of fill elements
>
> the size limits on the Sharefile system have been increased.

The GUI system has been enhanced substantially with additions like the **noredraw** property to reduce the number of redraws (and hence screen flicker). It should be used carefully as it can be confusing, it is not always sufficient to Paint the object itself one needs to Paint the parent instead which in itself can increase the scope of the flicker. It can take a while to become comfortable with this feature but it is worth persevering. It is a tidying up of the system, as it was possible to achieve the same effect with `⎕wcall` to change the Windows redraw facility directly.

```
  x ← ⎕wcall'SendMessage'(obj ⎕wi 'hwnd')'WM_SETREDRAW' (×flag) 0
```

Where flag is the inverse of the setting of the noredraw property and obj is the name of the object affected.

The new equivalent would be

```
  x ← obj ⎕wi 'noredraw' (~×flag)
```

which is much neater and now uses the standard interface `⎕wi` and not `⎕wcall`.

However, the example above does show how easy it is to access the underlying Windows API to produce effects that aren't defined in the standard interface.

Small but extremely useful changes, such as the inclusion of **Set** and **Ref** to the system object now allow the chaining of property sets and references on the system object. This brings it into line with the other GUI objects.

The GUI interface has been brought into line with other enhancements made by Microsoft to the basic objects, for example, images are now allowed on menus, in lists and combo boxes.

Printer and context help improvements

The control of the printer has been transformed and APL+Win now gives total control over the default printer and any other printers available either locally or over the network. The printer object was always the weakest part of the application but over the last few releases APL2000 has addressed this problem and in this release have completed the task. The printer object is now easy and straightforward to use.

The new context help is a welcome enhancement but the claimed interface with the Windows Help systems is only partially implemented. The old Winhelp version has the required definitions in the ADF file but the later HtmlHelp definitions have been omitted. To use the HtmlHelp system it is necessary to add the following definitions to the aplw.ini file (or to the adf file by recompiling it – full instructions are in the manuals).

The main entries for the ini file are listed below, the constants were lifted from the .h file provided with the Html Helpfile workshop. I haven't tried using this extensively yet as I am still using the old Winhelp version but I have checked that the HtmlHelp interface is called successfully once these entries are added.

```
[Call]
HtmlHelp=U(HW hwndCaller, *C pszFile, U uCommand, D dwData) LIB hhctrl.ocx
  ALIAS HtmlHelpA
[Constant]
HH_DISPLAY_TOPIC=              0x0000
HH_HELP_FINDER=                0x0000
HH_DISPLAY_TOC=                0x0001
HH_DISPLAY_INDEX=              0x0002
HH_DISPLAY_SEARCH=             0x0003
HH_SET_WIN_TYPE=               0x0004
HH_GET_WIN_TYPE=               0x0005
HH_GET_WIN_HANDLE=             0x0006
HH_ENUM_INFO_TYPE=             0x0007
HH_SET_INFO_TYPE=              0x0008
HH_SYNC=                       0x0009
HH_KEYWORD_LOOKUP=             0x000D
HH_DISPLAY_TEXT_POPUP=         0x000E
HH_HELP_CONTEXT=               0x000F
HH_TP_HELP_CONTEXTMENU=        0x0010
HH_TP_HELP_WM_HELP=            0x0011
HH_CLOSE_ALL=                  0x0012
HH_ALINK_LOOKUP=               0x0013
HH_GET_LAST_ERROR=             0x0014
HH_ENUM_CATEGORY=              0x0015
HH_ENUM_CATEGORY_IT=           0x0016
HH_RESET_IT_FILTER=            0x0017
HH_SET_INCLUSIVE_FILTER=       0x0018
HH_SET_EXCLUSIVE_FILTER=       0x0019
HH_INITIALIZE=                 0x001C
HH_UNINITIALIZE=               0x001D
HH_PRETRANSLATEMESSAGE=        0x00fd
HH_SET_GLOBAL_PROPERTY=        0x00fc
```

The usage of HtmlHelp() and the constants above is documented in the API reference that comes with the free HTML workshop from Microsoft at

> Http://msdn.microsoft.com/library/default.asp?url=/library
> /en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp

The main feature of this release is the Grid and much is rightly being made of this by APL2000. However I think the general improvement and polish that has been given to the product is equally important. The packaging of a runtime application is simple. Multi-threading/multi processor use can be introduced by simply using APL+Win as COM objects along side the normal APL workspace.

The clear implementation of the TCP/IP interface (using system function ⎕ni) allows:

> the linking of machines over a network
>
> the linking of tasks on the same machine
>
> direct access to the Internet
>
> the building of asp style Web pages

The use of Excel and other Active Object applications has been made simpler with the adoption of the standard Microsoft VB interface in addition to the original method. This means that any VB book can be used to navigate the standard published models. This has made it much easier to utilise all of the office applications. There are some example workspaces and examples in the documentation which show how Excel and APL+Win can be used together with either as a server to the other.

The understated and easily missed events **onNew** and **onAction** allow new User defined objects to be created and manipulated. This facility is extremely powerful and aids the system implementation by giving the developer the opportunity to define their own objects, properties and methods. A user defined object behaves in the same manner as a system object. This aids the reuse of code and can simplify the application development process. This facility has introduced a little instability if misused but once the code is right it behaves correctly.

Summary

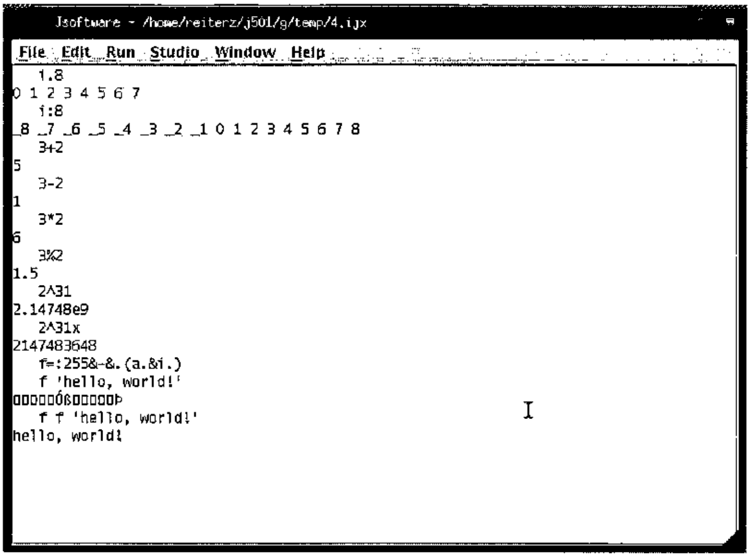
All of the enhancements, particularly the grid, make this a new product or upgrade worth buying. I can certainly recommend it.

J 5.01 Linux Beta h

reviewed by Zach Reiter

For those who have not seen the announcements on the J Forum, J for Linux finally has a GUI (Graphical User Interface). The new J 5.01 Beta h [1] available for Linux has a Java GUI. Of course, this is a beta and many details are likely to change before you read this. However, it still seems worthwhile to look ahead to the dramatic changes coming. Because J 5.01 Beta h will be outdated by the time you read this, please watch the J Forum and/or JSoftware website for updates. This new Java GUI, while currently only available for Linux, will eventually be available on all J platforms, extending J's cross-platform commitment to the GUI and to GUI programming.

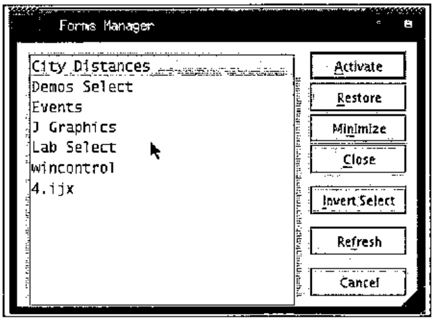
The J interface is quite different from the interface J for Windows users are accustomed to. The most noticeable change is the change from the MDI to SDI (Multiple/Single Document Interface), as can be seen below. This change allows for more complicated window arrangements.


Jsoftware - /home/reiterz/j501/g/temp/4.ijx
File Edit Run Studio Window Help

```
   1.8
0 1 2 3 4 5 6 7
   i:8
_8 _7 _6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6 7 8
   3+2
5
   3-2
1
   3*2
6
   3%2
1.5
   2^31
2.14748e9
   2^31x
2147483648
   f=:255&-&.(a.&i.)
   f 'hello, world!'
□□□□□Óß□□□□□Þ
   f f 'hello, world!'
hello, world!
```

Basic J commands listed here as executed in the session manager of the new beta.

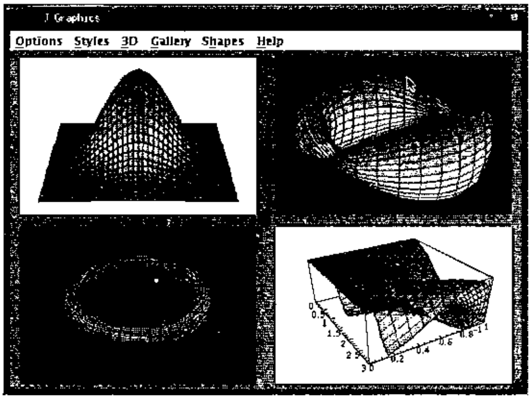
The menus generally resemble those of the Windows, with most of the changes due to the unavailability of features. Most strikingly, there are no Cut/Copy/ Paste options under Edit. These are nowhere to be found in the menubar, although Ctrl+X/C/V work as expected. Traditional X-style copying and pasting is fully present; when selecting, the selection is not copied until it is unselected, rather than the expected continual re-copying. This can then be pasted using the middle mouse button, either into J or into another X-compatible application. The Window menu does not list the windows as is common under MDI applications, but instead there is a Forms Manager, which is a quite effective replacement.


Forms Manager

City Distances
Demos Select
Events
J Graphics
Lab Select
wincontrol
4.ijx

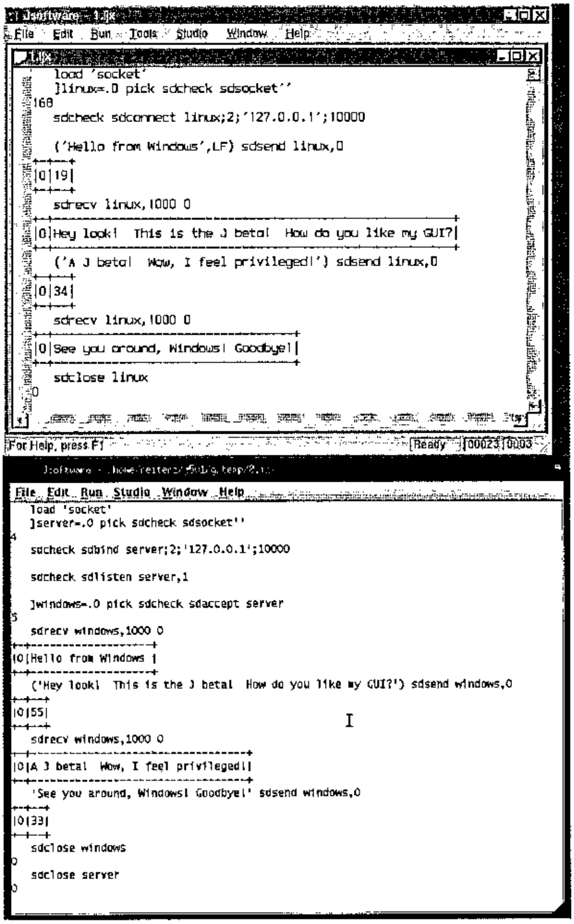
Not only does the Forms Manager list the edit windows that are open (as the Window menu did), it also lists all of the user created windows. From here windows can be minimized, restored, or closed. The Studio menu still lists the labs and demos options. The Lab Author is currently missing, as of beta h.

J still presents much of its full set of features. Many features have been rolled over from both J 4.05 for Linux and J 4.06 for Windows. OpenGL is not yet supported, but plot functions quite smoothly.



Plot's surface sampler, seen here under the beta

Printing was recently added to beta's list of features, it runs as expected (I now have a copy of the 'Surface of Revolution' from plot sitting on my desk). Memory mapped files continue to function as implemented under J 4.05 Linux. Sockets continue to function, and through the use of WINE [2], it is possible to have J 4.06 for Windows and J 5.01 beta h communicate on the same machine (see below).

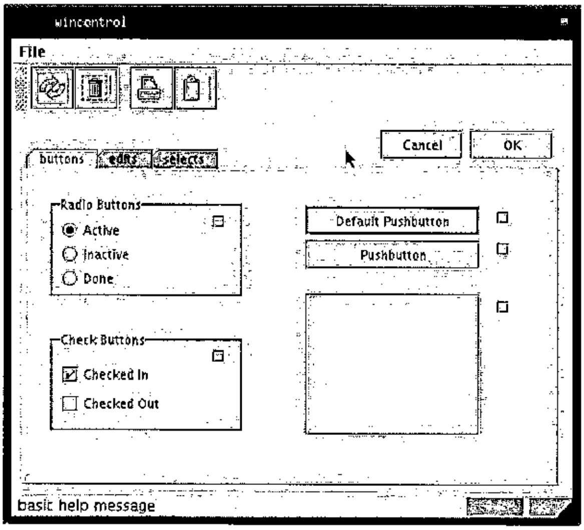

Jsoftware - 1.ijx
File Edit Run Tools Studio Window Help
1.ijx
For Help, press F1
Ready 00023 0003
File Edit Run Studio Window Help

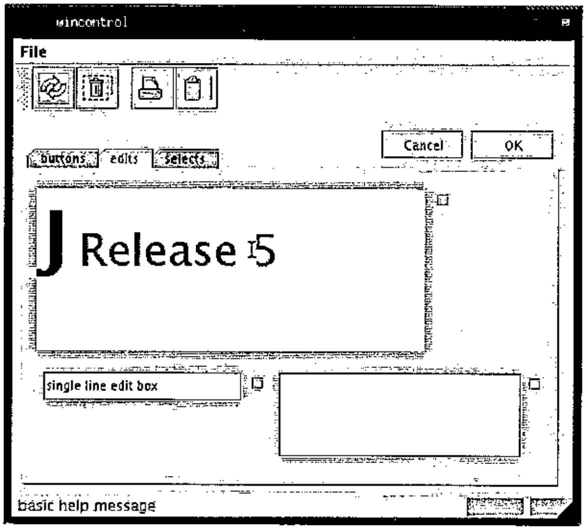
```
   load 'socket'
   ]linux=.0 pick sdcheck sdsocket''
168
   sdcheck sdconnect linux;2;'127.0.0.1';10000

   ('Hello from Windows',LF) sdsend linux,0
+-+--+
|0|19|
+-+--+
   sdrecv linux,1000 0
+-+--------------------------------------------------+
|0|Hey look!  This is the J beta!  How do you like my GUI?|
+-+--------------------------------------------------+
   ('A J beta!  Wow, I feel privileged!') sdsend linux,0
+-+--+
|0|34|
+-+--+
   sdrecv linux,1000 0
+-+--------------------------------+
|0|See you around, Windows! Goodbye!|
+-+--------------------------------+
   sdclose linux
0
```

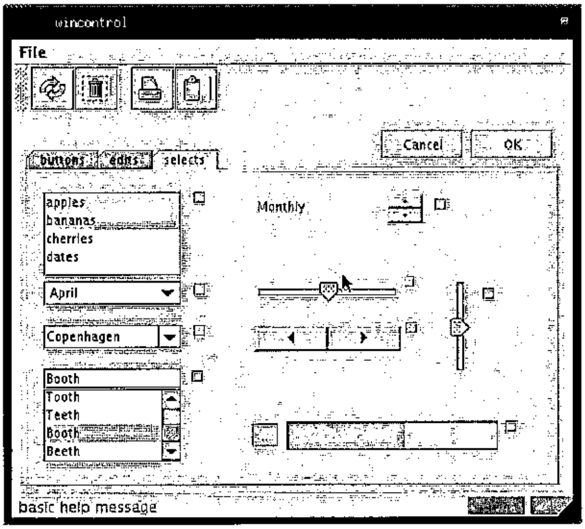
```
   load 'socket'
   ]server=.0 pick sdcheck sdsocket''
4
   sdcheck sdbind server;2;'127.0.0.1';10000

   sdcheck sdlisten server,1

   ]windows=.0 pick sdcheck sdaccept server
5
   sdrecv windows,1000 0
+-+-------------------+
|0|Hello from Windows |
+-+-------------------+
   ('Hey look!  This is the J beta!  How do you like my GUI?') sdsend windows,0
+-+--+
|0|55|
+-+--+
   sdrecv windows,1000 0
+-+---------------------------------+
|0|A J beta!  Wow, I feel privileged!|
+-+---------------------------------+
   'See you around, Windows! Goodbye!' sdsend windows,0
+-+--+
|0|33|
+-+--+
   sdclose windows
0
   sdclose server
0
```

Sockets communication between J for Windows and J for Linux. This accomplished under Linux, using WINE.

J offers a large set of its GUI controls; these can be seen in the controls demo, as shown below.



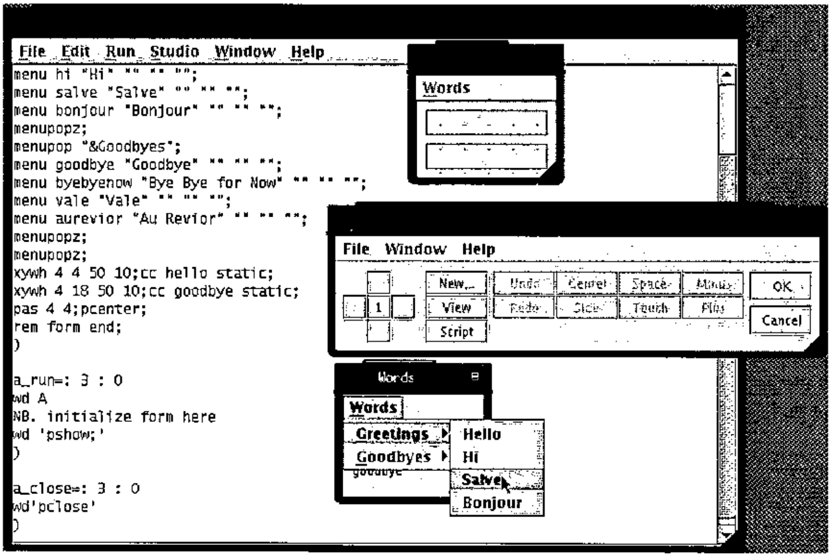




The Controls demo, which demonstrates the functionality of many of the controls J offers.

This beta includes the form editor and console. The form editor closely resembles that of Windows; unfortunately, it is painfully slower at the moment. The menu editor functions as expected, toolbars are not supported, but status bars are. Dragging controls is virtually impossible, because the screen is redrawn so infrequently. Resizing controls does not appear to function at the moment. Nonetheless, it provides a tool for composing the basics of a form; the details are easily worked out in the script window. The console remains familiar; J 4.05 Linux users will feel at home. The profile.ijs for beta h reveals much greater planned and implemented cross-platform support; it is reasonable to expect that shortly these details will no longer be left to the programmer.

However, as can be both expected and tolerated with a beta level product, there are some areas which need to be smoothed out. Scripts written for Linux will require some modifications to run smoothly under the beta; scripts written for Windows will require some slightly more significant modifications to run smoothly. In transferring from Windows to Linux, there many other issues, many of which are independent of J. There is no isipicture control under the beta, as there is under the J 4.06 Windows GUI. There are a number of minor GUI nuisances which persist: a freshly opened file always appears in the upper-left corner of the screen and is always scrolled all the way to the bottom; selecting run->window on the IJX window, runs it (which is probably NOT what you meant to do...).



```
menu hi "Hi" "" "" "";
menu salve "Salve" "" "" "";
menu bonjour "Bonjour" "" "" "";
menupopz;
menupop "&Goodbyes";
menu goodbye "Goodbye" "" "" "";
menu byebyenow "Bye Bye for Now" "" "" "";
menu vale "Vale" "" "" "";
menu aurevior "Au Revior" "" "" "";
menupopz;
menupopz;
xywh 4 4 50 10;cc hello static;
xywh 4 18 50 10;cc goodbye static;
pas 4 4;pcenter;
rem form end;
)

a_run=: 3 : 0
wd A
NB. initialize form here
wd 'pshow;'
)

a_close=: 3 : 0
wd'pclose'
)
```

The form editor; the script in the background is the J code; the highlighted Words window is a running example of a created form; the other two windows are the output of the form editor

J 5.01 Beta h is just that, a beta. There are many features which do not function at 100%; there are many features which do. I think it is important to recall that many of the features currently missing were added a few at a time, over the course of the 4.0x series. Iverson Software is continuing their tradition of excellence; the release of J 5.01 stable is definitely an event to look forward to.

References:

[1] JSoftware, http://www.jsoftware.com

[2] WINE, http://www.winehq.org/

How to Draw a Straight Line by A B Kempe

reviewed by Anthony Camacho

Have you ever encountered such an intriguing title? I encountered it in 1957 in *Mathematical Models* by Cundy and Rollett (OUP) and, ever since then I have kept an eye open for it. Its subtitle is A LECTURE ON LINKAGES, it is fifty-one pages long and it was published by Macmillan in 1877. Nice to see Vector is up to date!

I tried to get a copy from the national lending library, but failed and when I mentioned it to my son-in-law he downloaded it from Cornell University! It was worth the wait. I have nothing but praise for the book. Here is how Kempe begins:

> The great geometrician Euclid, before demonstrating to us the various propositions contained in his *Elements of Geometry*, requires that we should be able to effect certain processes. These *Postulates*, as the processes are termed, may roughly be said to demand that we should be able to describe straight lines and circles. And so great is the veneration that is paid to the master-geometrician, that there are many who would refuse the designation of "geometrical" to a demonstration which requires any other construction than can be effected by straight lines and circles. Hence many problems - such as, for example, the trisection of an angle - which can readily be effected by employing other simple means, are said to have no geometrical solution, since they cannot be achieved by straight lines and circles only.

But how do you draw a straight line? A circle is easy. In principle the method is perfect although in practice your pencil may be blunt. Creating a straight edge is as fraught with trial and error as making an engineer's surface table. How odd that Euclid didn't notice how different it is from a circle. Until 1874 no-one in England knew of a method for drawing a straight line that was, in principle, perfect. The first solution was found by a French army officer called Peaucellier and was brought to England by Professor Sylvester (the same J J Sylvester you have heard Donald McIntyre speak about) in a lecture at the Royal Institution in January 1874. Kempe treats us to descriptions of Watt's Parallel Motion, better linkages by Richard Roberts of Manchester and "Professor Tchebicheff" of St Petersburg, then M. Peaucellier's perfect solution with seven links and Mr Hart's (of Woolwich Academy) improved version with five links. Kempe's note on Watt's Parallel Motion is worth quoting in full.

I have been more than once asked to try and get rid of the objectionable term "parallel motion." I do not know how it came to be employed, and it certainly does not express what is intended. The apparatus does not give "parallel motion," but approximate "rectilinear motion." The expression, however has become crystallised, and I for one cannot undertake to find a solvent.

Kempe tells of his correspondence with Sylvester and their invention of the "Quadruplane" and describes other linkages, including those for dividing an angle into any desired number of equal parts. Beautifully written and fascinating!

How to find it: Google found it straight away, given the title in double quotes. *http://moa.cit.cornell.edu/math_K.html* takes you to the index by authors, section K, where it is easy to find Alfred Bray Kempe. The address of the first page of download was:

http://moa.cit.cornell.edu/Hunter/hunter.pl?
handle=cornell.library.math.Kemp009&id5

GENERAL ARTICLES

This section of Vector has articles of general interest, usually with relatively little in the way of computer code or complex algorithms. Our hope is that this section can be read with enjoyment by anyone who happens to pick up the magazine, even if they have never heard of APL or J.

APL in Commercial Systems Development: a Survey

by Stephen Taylor (sjt@lambenttechnology.com)

We planned to find out what use is being made of APL for commercial software development. Is commercial APL use prospering—or disappearing?

Are we writing new commercial systems in APL? Are we extending existing commercial systems to meet new requirements? Or are we mostly nursing APL systems into their old age until they can be replaced? If there is new work, in which countries and industries is it being done?

Put another way, we want to see the predictable future of APL (and its offspring) in commercial software development. Not what necessarily *will* happen, but what looks probable.

Questions

To learn that, we need to know the following about each system *you* know about:

Dialect Perhaps we mean 'language': we include not only dialects of APL but also A+, J, K and Niall.

Country Where is (most of) the development work currently being done?

Application For what is the software used?

System identification Owner or vendor, and name or number of the system/s. (You might want to tell us about a group of systems without distinguishing between them.) We will publish none of these, but use them only to ensure we don't count a system twice.

Developers How many people are working on the system's code?

Users How many people make use of the system, either through the application's own user interface or as a server?

Conception In what year did development work start?

Age Not the system's age in years, but the current part of its life cycle. This is characterised by the work being done, for which we have adapted Shakespeare's "seven ages of man":

> *Infancy* The system is being developed and is not yet supporting commercial activities of its intended users.
>
> *Childhood* The system is supporting commercial activities of at least some of its users, though there might still be teething problems.
>
> *Adolescence* Developers continue to add new function, either to complete the original specification, or to meet new requirements.
>
> *Maturity* From time to time developers work on the system to add requested functions, or to adapt it to changes in the environment.
>
> *Old Age* No new function is contemplated. Developers work on the system only to fix operating problems, or to adapt it to changes in the technical or regulatory environment.
>
> *Life Support* As for Old Age, but money is being spent now on replacing the system.

We've distinguished Life Support precisely. Organisations have often designated APL a 'non-strategic technology', intending to replace their APL systems as soon as convenient; which then often fails to happen. Many APL systems have spent long years wearing such nooses round their necks.

This reflects two things. One is the success of evangelists for one technology or another in eliciting commitment to corporate technology strategies. The other is APL's astonishing productivity. Business users with awkward needs who have secured a useful system are unenthusiastic about surrendering it when its replacement may be expensive or less satisfactory or both. Many business users have conducted long and successful defences of their APL systems against a corporate technology monoculture.

So we'll distinguish as moribund only systems where money is being spent now on their replacements.

Results

We thought that an informal survey would answer these questions: that we or our personal contacts would know most of what is going on. That has turned out not be so. (Or perhaps we just hope that there is much more to be found out.)

So we're summarising here what we've learned so far, and are asking *you* for more information.

Brazil

A consultant is using APL for statistical analysis and mapping applications.

Canada

A developer has told us of four customer systems in active use: one for economics consulting; the other three are packages sold to the finance industry.

An international data vendor has two major APL production systems based in Canada, with thousands of users worldwide. Systems are mature; support is moving to London.

Denmark

A software vendor sells a financial planning package built in Dyalog APL, estimates 5-10,000 users. Mature.

Finland

A consultant in Finland sells three packaged systems to the power industry there, one written in J; the others J with APL and Excel front ends. Mature.

Italy

A software house in Italy employs 15 developers to work on a financial planning package. Adolescence.

Spain

A consultant uses J for financial markets analysis.

UK

A major pensions company has been using two systems (APL2 and Dyalog APL) for 6-8 years; both systems are in adolescence; each has 3 developers and 50-100 users.

A major food company has a number of mature applications (Dyalog APL) all but one of which is now on life-support. No new development is likely.

USA

A major Wall Street investment bank has been using A+ systems in New York and London since 1988. Systems are now on life support with developers fired or reassigned.

A major commercial finance institution has developed an inventory of 20-40 batch and interactive APL systems over the last 13 years, supporting about 500 users. Development has shifted from APL2 (20 systems) to Dyalog APL. New and mature systems.

A consultant uses J to organise and analyse his demographic data.

Notes

Professor Ulrich Küsters tells us that academic use of APL has been growing since 1994 at the Catholic University of Eichstätt in Germany, where he and about a dozen other economists and statisticians use it.

Next

Is that it? Or is there more going on? Please use the form on the Vector web site at http://www.vector.org.uk to add more entries following this pattern. We'll publish an update sometime after the Madrid conference.

A Creative Use of ⎕*WATCHPOINTS* to Investigate Large Heritage WS in APL+Win

by Ian Clark (ian.clark@cognos.com)

Many commercially valuable APL workspaces consist of a nucleus of 10-20 year old code, possibly written originally to run on a mainframe and ported many times since between quite differently structured interactive environments, most recently to an event-driven environment such as Windows. The tool to be demonstrated comes into its own when you encounter such code for the first time and become responsible for maintaining it. Maybe the workspace has been built to engineering standards, with well-chosen naming conventions that have been rigorously adhered-to, consisting of several non-overlapping well-defined and smallish groups of functions, each of which constitutes a black-box and which can in principle be copied into a fresh workspace and studied in isolation with representative test data. Some hopes!

Such a workspace may contain over 1000 functions, each having between 10 and 500 lines of code, with perhaps 500 (semi)global variables being created and modified at runtime, few of which may be present when the workspace is first loaded. Functions have side-effects not described in their comments (which are not dependably up-to-date), they only work in-situ, bathed in a history-dependent environment, and extracting the closure of the set of functions calling/called by a given function ends up with roughly half the total number of functions. The workspace itself has no reliable cleavage lines which remain the same from function to function. It is however 'lumpy'. Thus, two groups of names can be discerned, those all in uppercase such as XINTRCV2 and those in mixed case with names like fnProcIntContentsA or printMultiPageDoc. The former you surmise is the heritage stratum, the latter is (largely) the interface to Windows. But when it comes to subsequent extensions to the heritage nucleus, the programmer concerned chose names according to his or her own inclinations. Meaningful – to whom? Symbolic – so what's the naming scheme? Compliant – sez who?

The question to be answered is this: when I perform such-and-such an interaction, what happens internally? To be precise: what functions are executed, and what globals do they alter? Traditional tools are of little use in such a code-'soup', until you basically understand how the workspace is structured. Only then can you guess which function to set a ⎕*STOP* in, and so begin a line-by-line walkthrough in an attempt to understand how the application works. Even then you may find that superficially quite straightforward interactions execute hundreds of lines in dozens of functions, most of which serve only to reassure the programmer that the 'environment' or 'soup' of cached data in the form of globals is up-to-date, at least where it matters to the task being performed.

Most newcomers to a team write exploratory tools for their own use. Once they become broadly familiar with how it all works the tool comes to be needed less and less. The owner even forgets how to use it, and so is less inclined to use it even when it would be useful. So the effort is largely once-off and soon written-off.

I wrote this tool with other people in mind (which meant me too, a few months down the road). I carefully avoid saying 'written with ease-of-use in mind'. Every programmer writes with ease-of-use in mind. Just as every politician is on the side of good government and against corruption. More to the point, the development team anticipated further newcomers, I found the tool most useful in practice and so I felt I could justify the effort. Thinking of new tasks for it justified the effort of designing a robust, easy-to-extend structure. Years of suffering at the hands of other people's tools which left sticky footprints in my workspaces made me design it as a single function, working in a way which any APLer could understand. Years of leaning over people's shoulders and using their keyboards made me avoid delta and delta-underbar. It has no status globals, register entries or .INI entries. This tool is the APL counterpart of the Willys Jeep. You can maintain it with nothing but a screwdriver and a monkey wrench.

The function is: tkv (short for 'track variables'). It creates four global variables in your workspace:

- tkv_L – a list of variable names to be tracked.
- tkv_T – a table of trace information which tkv builds and maintains. You don't handle it directly: tkv will display it for you in a choice of formats.
- tkv_I – assignment counter
- tkv_TS – a snapshot of the timestamp ⎕*TS*, being the last time anything was added to tkv_T. It furnishes an identification of the display output, when you come to make a collection of these things.

I could have combined them into one nested global; I could have kept this cache of data elsewhere; I could have written it as a 'user command'... I didn't, because it traded simplicity and portability for little or no gain.

If by a remote chance the prefix: 'tkv' happens to cause a name clash in your workspace, you may select another prefix by replacing all instances of 'tkv' in the function code, including line [0].

Here's how to use it. Assume the target workspace has been loaded and is waiting for an interaction. You have access to the APL session log and you have copied in fn: tkv

Now, turning to the APL session window, enter:

```
      tkv ⎕nl 2
>>> watchpoints inactive
>>> tracking table emptied, size=428
>>> these reserved vars removed from list tkv_L: tkv_L tkv_T tkv_I
>>> these too-frequently assigned vars removed from list tkv_L: ⍙ELX ⍙LOG t
>>> watchpoints active on 422 vars
>>> resume manual interaction with form(s)...
```

The right argument used here is a list of all the vars currently in the workspace. This is placed in the global: tkv_L. Alternatively you can load or copy tkv_L from any source you like, or create it by hand. Note that if tkv discovers it in char matrix form it converts it to a vector of nested char vectors ('strings'). Or if you just want to use or re-use the existing tkv_L, provide an empty right argument like this:

```
      tkv''
>>> watchpoints inactive
>>> tracking table emptied, size=422
>>> watchpoints active on 422 vars
>>> resume manual interaction with form(s)...
```

Now perform the desired sample interaction with the application. Turning back to the APL session, to view the results enter:

```
      zi←tkv'i'
      )ed zi
```

This generates the following report as a char vector with embedded ⎕TCNL chars in a global variable, zi:

```
========================================
GLOBAL TRACKING TABLE at:  4 Oct 2001 11:39
========================================

>>> last callback was: 14
Common suffix="fmLex_Page_Toolbar[36]>"

PageNumber:              1:
PrintPageNum:            2: fmLex_ShowPage[21]
PrintPageMax:            3: fmLex_GetHead[33]fmLex_ShowHead[18]fmLex_ShowPage[25
]
WINFO:                   4: fmLex_ShowHead[22]fmLex_ShowPage[25]
ROWattrMAT:              5: fmLex_View[32]fmLex_ShowHead[36]fmLex_ShowPage[25]
PrintPageMax:            6: fmLex_GetHead[33]fmLex_ShowHead[57]fmLex_ShowPage[25
]
WINFO:                   7: fmLex_ShowHead[59]fmLex_ShowPage[25]
DATAsigns:               8: GENoutFSM[15]fmLex_ShowPage[28]
RPTattrMAT:              9: GENoutFSM[15]fmLex_ShowPage[28]
pagZERO:                10: fmLex_ShowPage[30]
rho:                    11: SetGridAttr[76]fmLex_ShowPage[37]
WINFO:                  12: fmLex_ShowNames[50]fmLex_ShowPage[42]
PrintHeader:            13: fmLex_ShowPage[57]
PrintFooter:            14: fmLex_ShowPage[58]
```

Each sequence-numbered entry in the option 'I' report contains an extract of ⎕SI as it was when the assignment was made. To reduce the bulk of the output, tkv recognizes if a repeated suffix is common to every entry and strips it. The text of the suffix is not lost, however, you can see it at the head of the report:

```
  Common suffix="fmLex_Page_Toolbar[36]>"
```

You might hazard a guess that the whole transaction was triggered by pressing a toolbar button, for which the callback fn happened to be: `fmLex_Page_Toolbar` and all the work was done in line [36]. Notice that the data for the first assignment which took place (to var: `PageNumber`) contains nothing but the suffix, so the entry itself is empty.

The number in column 2 shows the order in which assignments occurred. A familiar feature of APL+ takes you directly to the line concerned when you double-click on column 3, say upon: `fmLex_ShowHead[22]` in the entry 7: for `WINFO` which happens to be:

```
        WINFO[1 2 3]←winfo[1 2 3]
```

Alternatively you might want to see the same data in the same order as your original list of variables, tkv_L. In the APL session, enter:

```
      zc←tkv'c'
      )ed zc
```

which generates the following report in zc:

```
========================================
GLOBAL TRACKING TABLE at:  4 Oct 2001 11:39
========================================

>>> last callback was: 14

DATAsigns
     8: GENoutFSM[15]fmLex_ShowPage[28]fmLex_Page_Toolbar[36]>
PageNumber
     1: fmLex_Page_Toolbar[36]>
PrintFooter
    14: fmLex_ShowPage[58]fmLex_Page_Toolbar[36]>
PrintHeader
    13: fmLex_ShowPage[57]fmLex_Page_Toolbar[36]>
PrintPageMax
     3: fmLex_GetHead[33]fmLex_ShowHead[18]fmLex_ShowPage[25]fmLex_Page_Toolbar[36]>
     6: fmLex_GetHead[33]fmLex_ShowHead[57]fmLex_ShowPage[25]fmLex_Page_Toolbar[36]>
PrintPageNum
     2: fmLex_ShowPage[21]fmLex_Page_Toolbar[36]>
ROWattrMAT
     5: fmLex_View[32]fmLex_ShowHead[36]fmLex_ShowPage[25]fmLex_Page_Toolbar[36]>
RPTattrMAT
     9: GENoutFSM[15]fmLex_ShowPage[28]fmLex_Page_Toolbar[36]>
WINFO
     4: fmLex_ShowHead[22]fmLex_ShowPage[25]fmLex_Page_Toolbar[36]>
     7: fmLex_ShowHead[59]fmLex_ShowPage[25]fmLex_Page_Toolbar[36]>
    12: fmLex_ShowNames[50]fmLex_ShowPage[42]fmLex_Page_Toolbar[36]>
pagZERO
    10: fmLex_ShowPage[30]fmLex_Page_Toolbar[36]>
rho
    11: SetGridAttr[76]fmLex_ShowPage[37]fmLex_Page_Toolbar[36]>
```

This table (option 'C') contains the same information as the previous one (option 'I'), but sorted by global name instead of by the order of assignments.

If you want to see the whole tracking table, whether or not a given variable was assigned, then use option 'S' instead of 'C'. Unlike options 'C' and 'I' which only show entries for variables actually assigned, this table has an entry for every tracked variable. With a large number of tracked variables, option 'S' isn't quite so useful as option 'C' since most will be blind entries, like the first 45 below (not all shown – the list has 463 entries so it has been snipped):

```
=========================================
GLOBAL TRACKING TABLE at:  4 Oct 2001 11:39
=========================================

>>> last callback was: 14

ACCESS_MANAGER
ACCTnrs
ACTcalctype
ACTcommon
ACTdim
...
DATAdir
DATAnumber
DATAsigns
    8: GENoutFSM[15]fmLex_ShowPage[28]fmLex_Page_Toolbar[36]>
DATTN
DAYPER
DBFM
DBI
...
```

I must say I don't use 'S' much. But it has a forensic value: if a variable isn't listed then it was not in table tkv_L and so was not being tracked. If a variable is missing from the option 'C' or 'I' tables however, it might also be because it just wasn't assigned.

What we see from all three lists (notice that their titles all carry the same timestamp, contained in tkv_TS) is that the sample interaction made 14 assignments to a total of 11 globals out of the 463 in the tracking list. We can see just this summary information by option 'O' (alias: 'o' or '0'):

```
      tkv'o'
>>> tracking table size=463
>>>  globals unassigned=452
>>>          " assigned=11
>>> last callback was: 14
```

The tool is meant for tracking globals only, although it can in principle track locals. It explicitly filters out assignments to locals, or more importantly localized versions of a given global, since such assignments confound the global being tracked. This feature (which inspects ⎕SINL) must be turned off (using a code switch) if you want to track all assignments to a given name, whether local, global or semiglobal. In practice this isn't a very useful thing to do, since (say) z in one fn bears little or no relation to z in another. Besides which the z entry grows enormous and contains little or no useful information, unless you already know a lot about the workspace (like where z is used).

How does it work? By assigning to ⎕*WATCHPOINTS* a 2-column array of which the first column is tkv_L, the list of variables, and the second column is a list of calls to tkv itself, the call at row n having argument: n. Thus, in our example, the first two rows are:

```
]rr ⎕watchpoints[1;]
'ACCESS_MANAGER' 'tkv 1'
]rr ⎕watchpoints[2;]
'ACCTnrs' 'tkv 2'
```

Whenever the variable named in row n gets created or reassigned, tkv runs with argument n, appending a formatted extract of ⎕*SI* to the nth element of tkv_T, thus building up the tracking table as the code executes. Note that ⎕*WATCHPOINTS* does not trigger a callback when a watched var is erased, but *does* trigger one upon an assignment that does not alter the value, e.g:

```
      WINFO←⌽⌽WINFO
```

Function: tkv is to be found inside a component file called TKV.sf, downloadable from the APL2000 website (http://www.apl2000.com/). Move TKV.sf into the folder of (say) *)LIB 1* and access it like so:

```
      ]ufile 1 tkv
Now using file 1 TKV
      ]uload tkv /r
1 object loaded
```

You can then see instructions using the conventional *]TKV ?*, or like this:

```
      tkv'?'
>>> RECOMMENDED USAGE:
 tkv ⎕NL 2 ⍝--start tracking all vars present (or provide your list)

>>> now resume interacting with the application, then...
 tkv 'S'   ⍝--turn off tracking and see the results

>>> To start a fresh tracking run...
 tkv ''    ⍝--uses existing list of vars: tkv_L, else like: tkv ' '
 tkv ' '   ⍝--recreates list: tkv_L from all vars present
 tkv ']'   ⍝--]ULOADs list: tkv_L

>>> now do the target action, then...
 tkv 'S'   ⍝--see the result, sorted by var
 tkv 'C'   ⍝--like 'S', but unassigned vars omitted
 tkv 'I'   ⍝--the result, sorted by order of assignment

>>> If and as required...
 tkv '+'   ⍝--turns ON tracking
 tkv '-'   ⍝--turns OFF tracking
 tkv 'O'   ⍝--summarizes tracking table
 tkv 'T'   ⍝--empties tracking table
```

```
>>> tkv creates these globals (erase after use):
 tkv_I     ⍝--tracking counter: counts assignments to sequence them
 tkv_L     ⍝--list of vars tracked (cmx or strings)
 tkv_T     ⍝--assignments table of vars in tkv_L (strings)
 tkv_TS    ⍝--⎕TS at latest callback of tkv
 )ERASE tkv_I tkv_L tkv_T tkv_TS  tvk

z←tkv_larg tkv tkv_n;i;j;tkv_title;zz;⎕ELX;⎕IO
⍝track the appearance of global vars.
⍝∇ IAC 05Oct2001
⍝mydev
 ⎕ELX←'⎕DM'
 ⎕IO←1 ⍝--sometimes this has changed locally when tkv is called
 z←θ
 tkv_title←''

⍝---------------------
⍝ RT ARG OPTIONS:
⍝ '?' ⍝--shows usage
⍝ '' ' ' ']' 'S' 'I' 'C' 'O' '-' '+' ...see :CASE '?' below.
⍝
⍝(These are for internal use only)...
⍝ 'L'        --(re)create the tracking var list
⍝ 'T'        --reset the tracking table
⍝ 'H'        --output table header
⍝ '⍳'        --callback count message
⍝ 'O'        --summarise tkv_T
⍝ '-'        --remove reserved names from tkv_L
⍝ '↑'        --check tkv_L converted from cmx to (strings)
⍝ n          --watchpoint callback, n ∊ ⍳⍴tkv_L
⍝---------------------

 :If 20<↑⍴⎕SI ⍝--runaway recursion safety net...
     0 0⍴ HALT ⎕stop 'tkv'
HALT: ⍝--halts here if )SI gone too deep
 :EndIf

⍝INTERNAL UTILITIES (LEFT ARG∊ 1 2 3)...
 :If 2=⎕NC'tkv_larg'
    ⍝ ...Rt arg: tkv_n is now some value to be processed
     :Select tkv_larg
     :Case 1 ⍝--delete trailing blanks
         z←(⌽∨\⌽z≠' ')/z←tkv_n
         →0
     :Case 2 ⍝--return common suffix
         z←θ
         :For zz :In tkv_n
             :If 0<⍴zz
                 :If 0<⍴z
                     z←(⌽∧\⌽z=(-⍴z)↑zz)/z
                 :Else
                     z←zz
                 :EndIf
             :EndIf
```

```
          :EndFor
          z←(']'=↑z)↓z ⍝--discount a leading ']'
          →0
      :Case 3 ⍝--reformat timestamp: tkv_n to: dd Mmm yyyy hh:mm
          zz←tkv_n
          z←2 0⍕zz[3]
          z←z,' ',(12 3⍴'JanFebMarAprMayJunJulAugSepOctNovDec')[zz[2];]
          z←z,' ',⍕zz[1]
          z←z,' ',¯2↑⍕100+zz[4]
          z←z,':',¯2↑⍕100+zz[5]
          →0
      :Else ⍝-assume it's a title for reporting output
          tkv_title←,⍕tkv_larg ⍝...and don't exit...
      :Endselect
  :EndIf

 ⍝Check for complex rt arg...
  :If 2=⍴⍴tkv_n ⍝--place cmx as nested array in tkv_L, then start tracking
  :OrIf (1=⍴⍴tkv_n)∧(2=≡tkv_n) ⍝--place nested array in tkv_L, then start tra
 cking
      :If 2=⍴⍴tkv_n ⋄ tkv_n←(⎕SPLIT tkv_n)~¨' ' ⋄ :EndIf
      tkv_L←tkv_n
      z←tkv''
      →0
  :ElseIf (1=⍴⍴tkv_n)∧(0≠×/⍴tkv_n) ⍝--treat as repeated calls
      z←∊tkv¨tkv_n
      →0
  :EndIf



  :Select tkv_n

  ⍝---------------------
  :Case '?' ⍝--show usage
  ⍝---------------------
      ">>> RECOMMENDED USAGE:"
      " tkv ⎕NL 2 ⍝--start tracking all vars (or provide your own list)"
      ""
      ">>> now resume interacting with the application, then..."
      " tkv 'S'   ⍝--turn off tracking and see the results"
      ""
      ">>> To start a fresh tracking run..."
      " tkv ''    ⍝--uses existing list of vars: tkv_L, else like: tkv ' '"
      " tkv ' '   ⍝--recreates list: tkv_L from all vars present"
      " tkv ']'   ⍝--]ULOADs list: tkv_L"
      ""
      ">>> now do the target action, then..."
      " tkv 'S'   ⍝--see the result, sorted by var"
      " tkv 'C'   ⍝--like 'S', but unassigned vars omitted"
      " tkv 'I'   ⍝--the result, sorted by order of assignment"
      ""
      ">>> If and as required..."
      " tkv '+'   ⍝--turns ON tracking"
      " tkv '-'   ⍝--turns OFF tracking"
```

```
    " tkv 'O'   ⍝--summarizes tracking table"
    " tkv 'T'   ⍝--empties tracking table"
    ""
    ">>> tkv creates these globals (erase after use):"
    " tkv_I     ⍝--tracking counter: counts assignments to sequence them"
    " tkv_L     ⍝--list of vars tracked (cmx or strings)"
    " tkv_T     ⍝--assignments table of vars in tkv_L (strings)"
    " tkv_TS    ⍝--⎕TS at latest callback of tkv"
    " )ERASE tkv_I tkv_L tkv_T tkv_TS  tkv"

⍝---------------------
:CaseList ⊖ '' ' ' ']' ⍝--Inits, clears globs & runs OPENrtpg
⍝---------------------
    z←tkv'-'
   ⍝Decide what to do with (existing?) var table: tkv_L
    :If     tkv_n≡' ' ⋄ z←tkv'L'
    :ElseIf tkv_n≡']' ⋄ z←tkv'#'
    :EndIf

    z←tkv'T+'
    '>>> resume manual interaction with form(s)...'

    z←0 0⍴0

⍝---------------------
:Case 'L' ⍝--regenerate the tracking list tkv_L from all vars found in ws
⍝---------------------
    tkv_L←((⎕SPLIT ⎕NL 2)~¨' ')
    z←tkv'~'

    z←0 0⍴0

⍝---------------------
:Case '#' ⍝--load the tracking list tkv_L
⍝---------------------
    :If 2=⎕NC'tkv_L' ⋄ zz←tkv_L ⋄ :Else ⋄ zz←0 ⋄ :EndIf
    ⎕UCMD'ULOAD tkv_L /REPLACE'

   ⍝How has tkv_L changed, if at all?
    :If 2=⎕NC'tkv_L'
        :If zz≡tkv_L
            '>>> var list found, size=',⍕⍴tkv_L
        :Else
            '>>> new var list loaded, size=',⍕⍴tkv_L
        :EndIf
    :Else
        z←tkv'L'
        '>>> no tracking list found, list regenerated, size=',⍕⍴tkv_L
    :EndIf

    z←0 0⍴0

⍝---------------------
:Case 'H' ⍝--table header
⍝---------------------
```

```
    z←'GLOBAL TRACKING TABLE at: ',3 tkv tkv_TS
    z←z,(0≠×/⍴tkv_title)/⎕TCNL,tkv_title
    i←(⍴z)⍴'=' ⋄ z←i,⎕TCNL,z,⎕TCNL,i,⎕TCNL
    z←z,tkv'⍳'

⍝---------------------
:Case '⍳' ⍝--callback count
⍝---------------------
    z←⎕TCNL,'>>> last callback was: ',(⍕tkv_I),⎕TCNL

⍝---------------------
:CaseList 's' 'S' '⎕' ⍝--show the table, sort by tkv_T
⍝---------------------
    →(0=tkv_I)/ERR0
    z←tkv'-' ⍝--turn off tracking
    z←tkv_title tkv'H'
    :For i :In ⍳↑⍴tkv_T
        z←z,⎕TCNL,⍕(i⊃tkv_L),(i⊃tkv_T)
    :EndFor

⍝---------------------
:CaseList 'c' 'C' ⍝--show the table, sort by tkv_T, omitting blind entries
⍝---------------------
    →(0=tkv_I)/ERR0
    z←tkv'-' ⍝--turn off tracking
    z←tkv_title tkv'H'
    :For i :In ⍳↑⍴tkv_T
        :If 0<⍴i⊃tkv_T
            z←z,⎕TCNL,⍕(i⊃tkv_L),(i⊃tkv_T)
        :EndIf
    :EndFor

⍝---------------------
:CaseList 'i' 'I' '⍞' ⍝--show the table, sort by tkv_I
⍝---------------------
    →(0=tkv_I)/ERR0
    z←tkv'-' ⍝--turn off tracking
    z←⍬
    j←⊃⊃⌈/⍴¨tkv_L ⍝--width of global name field
    :For i :In ⍳↑⍴tkv_T
        :If 0<×/⍴zz←i⊃tkv_T
            zz←(zz≠⎕TCNL)⊂zz
            zz←zz,¨⊂j↑(i⊃tkv_L),':'
            z←z,zz
        :EndIf
    :EndFor
    zz←⍎¨5↑¨z ⍝--counter vec (NB: change 5? Change 5 in line [206] too)
    z←z[⍋zz] ⍝--sort zz by counter field
    z←1 tkv¨(-j)⌽¨z ⍝--bring global name field to front & trim
    z←(-↑⍴zz←2 tkv z)↓¨z ⍝--drop common suffix zz from z
    z←(tkv_title tkv'H'),('Common suffix="',zz,'"',⎕TCNL),∊⎕TCNL,¨z

⍝---------------------
:Case '~' ⍝--remove tkv locals & globals from the tracking list tkv_L
⍝---------------------
```

```
    z←'zz' 'tkv_larg' 'tkv_n' 'tkv_L' 'tkv_T' 'tkv_I' 'tkv_TS' 'tkv_title',
,¨'ijz'
    :If 0≠⍴z←(z∊tkv_L)/z
        tkv_L←tkv_L~z
        '>>> these reserved vars removed from list tkv_L:',⍕z
    :EndIf

    :IF 1 ⍝--CODE SWITCH: set to 0 to omit removal of the following vars...
        z←'⍙ELX' '⍙LOG',,¨'cdrt'
        :If 0≠⍴z←(z∊tkv_L)/z
            tkv_L←tkv_L~z
            '>>> these too-frequently assigned vars removed from tkv_L:',⍕z
        :EndIf
    :ENDIF

⍝---------------------
:Case '↑' ⍝--ensure tkv_L is (strings) not cmx
⍝---------------------
    :If 2=⍴⍴tkv_L
        tkv_L←(⎕SPLIT tkv_L)~¨' '
    :EndIf

    z←0 0⍴0

⍝---------------------
:CaseList 'O' 'o' '0' ⍝--summarise the tracking table
⍝---------------------
    :If (⍴tkv_L)≠⍴tkv_T
        z←'>>> tracking table mismatch: ',⍕(⍴tkv_L)(⍴tkv_T)
    :Else
        z←tkv'⍴'
        z←z,⎕TCNL,'>>>  globals unassigned=',⍕+/0=∊⍴¨tkv_T
        z←z,⎕TCNL,'>>>            " assigned=',⍕+/0≠∊⍴¨tkv_T
        z←z,tkv'⍳'
    :EndIf

⍝---------------------
:Case '⍴' ⍝--tracking table size
⍝---------------------
    z←'>>> tracking table size=',⍕⍴tkv_T

⍝---------------------
:Case 'T' ⍝--reset the tracking table
⍝---------------------
    tkv_T←(⍴tkv_L)⍴⊂⍬ ⍝--create empty tracking table
    tkv_I←0 ⍝--start the tracking counter
    '>>> tracking table emptied, size=',⍕⍴tkv_T

    z←0 0⍴0

⍝---------------------
:Case '-' ⍝--turn watching OFF
⍝---------------------
    ⎕WATCHPOINTS←''
    '>>> watchpoints inactive'
```

```
      z←0 0⍴0

  ⍝---------------------
  :Case '+' ⍝--turn watching ON
  ⍝---------------------
      z←tkv'+-'
      ⎕WATCHPOINTS←⍉tkv_L,[.5] (⊂'tkv '),¨⍕¨⍳⍴tkv_L
      '>>> watchpoints active on ',(⍕⍴tkv_L),' vars'

      z←0 0⍴0

  ⍝---------------------
  :CaseList ⍳⍴tkv_T ⍝--all the valid callbacks
  ⍝---------------------
      :If ~(⊂,tkv_n⊃tkv_L)∊(z≠' ')⊂z←,' ',⎕SINL ⍝--if var is not local...
     ⍝:If 1 ⍝--whether var is localised or not...
     ⍝CODE SWITCH: ...decomment one or other of the above :If-statements
          z←tkv_n⊃tkv_L              ⍝ the var name
        ⍝z,' ← ',(⎕UCMD'RR ',z)      ⍝ trace the assignment in Session Window
     ⍝CODE SWITCH: ...decomment the above line to trace in Session Window
     ⍝               ...this needs the ucmd: ]RR to be installed
          z←tkv_n⊃tkv_T              ⍝ get the tracking table entry
          z←z,⎕TCNL                  ⍝ callback entry separator
          z←z,5 0⍕tkv_I←tkv_I+1      ⍝ callback counter prefixed
          z←z,': '                   ⍝ callback counter terminator
          i←(,2 0↓⎕SI)~' '           ⍝ omit ⎕SI[1 2;] (tkv[.] * ⎕WATCHPOINT[1]
)
          i←((i∊'<>')⍳1)↓i           ⍝ strip redundant on-event & event info
          z←z,i                      ⍝ append the list of fns
          (tkv_n⊃tkv_T)←z            ⍝ ...put z back into table
          tkv_TS←⎕TS                 ⍝ record the time
      :EndIf

      z←0 ⍝--⎕WATCHPOINT _needs_ the callback fn (tkv) to return 0
          ⍝  otherwise it suspends with VALUE ERROR !!

  ⍝---------------------
  :Else
  ⍝---------------------
      z←'>>> bad arg: ',⍕tkv_n

  :Endselect
  →0

 ERR0: '>>> no globals were assigned' ⋄ →0
```

TECHNICAL SECTION

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

Contents

TECHNICAL CORRESPONDENCE

Rank and Extending the Domains

From: Phil Chastney 6th June 2002

With reference to his letter in Vector 18.4, I thank Norman Thomson for his interest, but I wonder what his understanding of Occam's Razor is. If he sees it as no more than a word-count function, then the winner surely is Trenchard More, 25 or more years ago, whose definition, after a statement of the problem, also takes 16 words:

> Given an arbitrary monadic operation ?, how does one produce another operation that applies ? to the items of an array rather than the array itself? ... Let ? be transformed to the monadic operation : ? by an operator called the *replacement operator*.

The trouble with this definition, as with Norman's, is that it leaves the process undefined. There are infinitely many processes satisfying the stated requirements, and we need something more specific. The process described in my paper accommodates arrays of instances of parameterised classes and is, I think, the smallest process to do so adequately.

From: Howard A. Peelle (hapeelle@educ.umass.edu)

In the previous issue of Vector 18.4, Norman Thomson expounded on the virtues of understanding noun and verb rank in J. To his illustrations (responding to Phil Chastney's APL examples on "Extending the Domain" in Vector 18.3), please add the following punch lines.

```
    [ t =: >:i.3 4       NB. Same t is Increment of axis Integers
                        in a 3 by 4 table
 1  2  3  4
 5  6  7  8
 9 10 11 12
```

Shift each row of the table:

```
    1 0 2 |."0 1 t       NB. Each Rank 0 atom on left Rotate
                        each corresponding Rank 1 list on right
  2  3  4  1
  5  6  7  8
 11 12  9 10

    [ u =: 2 3 $ 'abcdef'
     NB. Same u is 2 by 3 Shape characters
                        into a table
 abc
 def
```

Basic ways to join a table with a (short) list:

```
    u , 'XY'             NB. u Append list 'XY'
                        with blank fill element
 abc
 def
 XY

    u ,"1 'XY'                NB. u Append each Rank 1 list
                        to list 'XY'
 abcXY
 defXY

    u ,"1 0 'XY'              NB. u Append each Rank 1 list
                        to each Rank 0 atom of 'XY'
 abcX
 defY
```

(Or, using Stitch, simply: `u ,. 'XY'`)

```
    u ,: 'XY'            NB. u Laminate 'XY' with fill blanks
                        for 2 by 2 by 3 shape
 abc
 def

 XY
```

At Play With J: J Be Nimble, J Be Quick

by Eugene McDonnell

Nim Addition

Nim is a simple game that someone knowledgeable playing against someone naïve can almost always win. It was featured in the 1960s film *Last Year in Marienbad,* where two men in a bar play with a number of piles of matchsticks. The players, in turn, take any number of matchsticks from any one of the piles. The object is to be the player who takes the last match or matches. Its name came perhaps from *nimm,* the third person singular imperative of the German verb *nehmen,* meaning *to take.* The trick in Nim is knowing that a position is either safe or unsafe, depending on whether the Nim sum of the number of matches in each pile is or is not zero. The Nim sum can be obtained by converting the number of matchsticks in each pile to binary, and inserting not-equals, or exclusive-or over this, then converting back to integer. For example, if there are three piles, with three, five, and seven matches in the piles, the Nim sum is obtained in three steps. First, the binary forms of the numbers are taken:

```
   piles =: 3 5 7
   #: piles
0 1 1
1 0 1
1 1 1
```

The not-equal function yields the parity of its summands:

```
   ~: / #: piles
0 0 1
```

This is converted to decimal:

```
   #. ~: / #: piles
1
```

The function NS encapsulates this:

```
   NS =: ~: / &. #: NB. not-equal insert dual antibase
   NS piles
1
```

A safe move can be made if and only if the Nim sum of the piles is not zero, meaning unsafe. If a position is safe, any move will change it to unsafe. Furthermore, if the Nim sum of the piles is nonzero, it can always be made safe by subtracting from one of the piles. There will always be at least one such pile. For example, given the piles 3 5 7, with Nim sum 0 0 1, we can subtract one from any one of the piles. Thus, three different safe moves can be made, resulting in one of 2 5 7 or 3 4 7 or 3 5 6.

```
   NS/"1 [ 2 5 7, 3 4 7,: 3 5 6
0 0 0
```

The choice of which pile to subtract from, when more than one is a candidate, is arbitrary.

Now, suppose we have a Nim sum of a list of piles that is a bit more complicated (the function **h** displays the binary form of the piles and its binary sum):

```
   h =: ,. @ (#: ; [: ~:/ #:)
   h 10 11 4
+-------+
|1 0 1 0|
|1 0 1 1|
|0 1 0 0|
+-------+
|0 1 0 1|
+-------+
```

The only solution for this is to subtract 3 from the last pile, which yields 10 11 1:

```
   h 10 11 1
+-------+
|1 0 1 0|
|1 0 1 1|
|0 0 0 1|
+-------+
|0 0 0 0|
+-------+
```

There is a certain amount of art in playing a winning game of Nim.

Nim multiplication

John H. Conway and Richard K. Guy have written *The Book of Numbers*. I was encouraged to read this by Ken Iverson's Lab which uses J to explore many of the parts of this book. Its last chapter is "Infinite and Transcendental Numbers", and in it, to my surprise, is a discussion of the game of Nim. Conway & Guy coined the word *nimbers* for the ordinary decimal integers, in the Nim context. I think they are confusing the numbers involved with the functions used with them. Conway & Guy, in addition to discussing Nim addition, also treat Nim multiplication, which they state is valuable in studying the digital transmission of information, in particular "the integral lexicographical code of minimal distance 3". They give a multiplication table for the first sixteen nonnegative integers. They also write

> And here's all you need to know about the multiplication of nimbers:
>
> If the 'larger' of two different nimbers is 1 or 2 or 4 or 16 or 256 or 65536 or 4294967296 or ..., you multiply them just as you multiply the corresponding ordinary numbers. The product of one of these *special* nimbers with itself is obtained by taking 1½ times its ordinary value.

I found it impossible to use this rule for nimbers greater than 4. I turned to Google for help, and found that Sloane's *On Line Encyclopedia of Integer Sequences* contained entries on Nim multiplication which included a function which built a Nim multiplication table. The problem was that the function was written in Maple, and although I am able to read very simple Maple, this one used built-in functions with meanings I couldn't grasp, even after I found a Maple manual on the Web. After weeks of trying to come to terms with it, appealing for help to several people I thought could help, but didn't, I appealed for help to the J discussion group on the Web and also wrote appeals to Conway & Guy. Both pleas were successful; Mike Day read my appeal to the J group, was able to decipher the Maple, and turned it into J, and Professor Guy's return letter gave me examples showing more in detail how Nim multiplication was done. Here is Mike Day's function:

```
NB. Mike Day mt

nimsum =: ~:/&.#:@,"0/~    NB. EEmcD

sort =: /:~

nimtimes =: (< @: ,) { (mt @: >./ ) NB. exploit mt

NB. verb mt is a fairly close simulation of the maple source
NB. - not necessarily good J!
NB.
mt =: verb define
iN =. i. >: N =. y.
NB. ==============================================
NB. lines 1 to 6
MT =. 0 $~ 2 # N + 1    NB. initialise MT with 0 top & left
MT =. iN 1 } MT          NB. and indices in row 1
NB. MT =. iN 1 }"_1 MT  NB. originally also in col 1
NB. - We can defer symmetrising and just work on diag
NB. and upper triangle
NB. =======================================================
NB. lines 7 - 11 - should be able to cut out some loops
                   NB. by eg recursion or scan
for_a. 2 }. iN do.
 for_b. iN }.~ a do.
  t1 =. i. 0
  for_i. i. a do.
   for_j. i. b do.
NB. =======================================================
    NB. lines 12-24 are preamble to line 25
    NB. references to stored AT where available
    NB. or nimsum where not avail. obscures the process -
    NB. This is ok on a fast m/c and/or for small N

    NB. line 25 (26 is a comment) ...
    NB. sort refs since using diag and upper triangle only
    refs =. sort each (i,b);(a,j);(i,j)
    t1 =. t1 , nimsum / refs { MT
NB. =======================================================
   end.   NB. line 27
  end.    NB. line 28
NB. =======================================================
  NB. line 29 - seems to require the nub
  t2 =. sort ~. t1
NB. =======================================================
  NB. lines 31 - 36 - locate first element of t2
  NB. not equal to its index
  j =. 1 i.~ t2 ~: i. # t2
NB. =======================================================
  NB. line 37 only
  MT =. j (<a,b) } MT  NB. don't need line 38
NB. =======================================================
 end.     NB. line 39
end.      NB. line 40
NB. =======================================================

NB. extra line to symmetrise
MT + (iN >/ iN) * |: MT
)
```

This function is a faithful translation of the Maple program. Day made no pretense that this was good J. All credit is owed to him for enabling others to contribute. Ken Iverson made some revisions to Day's function and I added my own changes. Here is its latest manifestation:

```
mt=: monad define
iN=.i.N=.y.
MT=.(>.|:)iN 1}0$~,~N
for_a. 2}.iN do.for_b. a}.iN do.
 c=.a,b,|:(#:i.@*/)a,b
 r=.<"1[0 1|:(2 1,0 3,:2 3){c
 t=.~.(~:/&.#:)"1 r{"1 2 MT
 j=.(0:i.~]e.~[:i.2:+>./)t
 MT=.j((;|.)a,b)}MT
end.end.
)
```

The argument N=.y. is the size of the square desired. Nim multiplication is commutative so the derivation of one nondiagonal value allows its symmetrical twin to be created at the same time. I use the term *nonneg* in order to shorten the phrase *nonnegative integer*. The list iN of the first N nonnegs serves to initialize row 1 and column 1, and is also used to determine the values of the loop counters a and b. An NxN matrix of zeros is created (0$~,~N) and row one is amended with iN; column 1 is set by forming the maximum of this matrix and its transpose (>.|:). For N=5 the result is:

```
   y.=:5
   ]iN=.i.N=.y.
0 1 2 3 4
   ]MT=.(>.|:)iN 1}0$~,~N
0 0 0 0 0
0 1 2 3 4
0 2 0 0 0
0 3 0 0 0
0 4 0 0 0
```

Having handled rows and columns 1 and 2 thus easily, the next value we need to create is that in row 2, column 2. After that comes the item in row 2 column 3 and row 3 column 2, and so on until row 2 and column 2 is completed. Then comes 3 3 and 3 4 (and 4 3), and finally 4 4. The process for 2 2 is as follows:

```
   a=.b=.2
   |:(#:i.@*/)a,b
0 0 1 1
0 1 0 1
```

This matrix gets two new rows, a row of all a's on a row of all b's. The resulting four rows correspond to those labelled a, b, i and j in the Maple function. Combinations of these rows can be assembled so that critical values preceding the one currently being made can be used according to a rule which I can't explain, since I don't understand it.

```
   ]c=.a,b,|:(#:i.@*/)a,b
2 2 2 2
2 2 2 2
0 0 1 1
0 1 0 1
```

The actual selection uses items at `2 1 (i,b)`, `0 3 (a,j)`, and `2 3 (i,j)`.

```
   (2 1,0 3,:2 3){c
0 0 1 1
2 2 2 2

2 2 2 2
0 1 0 1

0 0 1 1
0 1 0 1
```

These are transposed by placing the first two axes at the end (`0 1|:`)

```
   0 1|:(2 1,0 3,:2 3){c
0 2
2 0
0 0

0 2
2 1
0 1

1 2
2 0
1 0

1 2
2 1
1 1
```

In order for these to be used as indices to MT, their rows are boxed:

```
      ]r=.<"1[0 1|:(2 1,0 3,:2 3){c
+---+---+---+
|0 2|2 0|0 0|
+---+---+---+
|0 2|2 1|0 1|
+---+---+---+
|1 2|2 0|1 0|
+---+---+---+
|1 2|2 1|1 1|
+---+---+---+
```

The table of indices selects the needed values: (`r{"1 2 MT`), and the Nim sums of the rows determined (`(~:/&.#:)"1`) and duplicate sums are removed (`~.`)

```
     r{"1 2 MT
0 0 0
0 2 0
2 0 0
2 2 1
     (~:/&.#:)"1 r{"1 2 MT
0 2 2 1
     t=.~.(~:/&.#:)"1 r{"1 2 MT
     t
0 2 1
```

The mysterious part comes now. The value `j` to be stored at (`a,b`) is the *least* nonneg not in `t`. Why this produces the Nim multiplication of `a` and `b` is beyond me to explain.

The candidates for j are all in the first `2+>./t` nonnegs:

```
     i.2+>./t
0 1 2 3
```

The ones already present in `t` are identified:

```
     t e.~ 0 1 2 3
1 1 1 0
```

and `j` is the index of the first zero in this list:

```
     0 i.~1 1 1 0
3
```

Here's the whole:

```
   ]j=.(0:i.~]e.~[:i.2:+>./)
3
```

and here is the finished 5×5 table:

```
   mt 5
0 0 0  0  0
0 1 2  3  4
0 2 3  1  8
0 3 1  2 12
0 4 8 12  6
```

We now know how to make a Nim multiplication table, and I wanted to know how efficient this function was. I found that the number of times t the inner j loop of Day's program was used, for differing sizes s of arguments, to be:

```
s   t
2   4
3  19
4  55
5 125
6 245
7 434
8 714
```

The fifth difference of t is zero, so a polynomial of degree 4 can be found:

```
   diff=:2: --/\ ]
   t =: 4 19 55 125 245 434 714
   diff t
15 36 70 120 189 280
   diff^:2 t
21 34 50 69 91
   diff^:3 t
13 16 19 22
   diff^:4 t
3 3 3
   diff^:5 t
0 0
```

The polynomial is formed like this:

```
   x=:i.#t
   t %. x^/i.5x
4 97r12 43r8 17r12 1r8
```

These can be made the numerators for a rational polynomial:

```
   ]c=:24 2 3 2 3*4 97 43 17 1
96 194 129 34 3
   polyn=: c&p.%24&p.
   polyn i.7
4 19 55 125 245 434 714
```

I'll make a slight detour here, to explore the result of `polyn t` further. I found that the fourth degree polynomial for the figurate numbers of order 5 is relevant. These numbers are those in the fifth diagonal of the Pascal triangle. In fact, I found that a multiple of these added to the figuarte numbers of order 4 gives us our numbers:

```
   ]p4=:3!3+i.7
1 4 10 20 35 56 84
   ]p5=:4!4+i.7
1 5 15 35 70 126 210
   p4+3*p5
4 19 55 125 245 434 714
```

The detour is over. Now I'll use our polynomial to find how often the inner loop is entered for a size 209 table:

```
   poly 209x
251673415
```

A quarter of a thousand million iterations seems excessive.

This makes clear how ridiculous and expensive it is to have to make a 209×209 table in order to get the Nim product of 167 and 208! The letter I got from Professor Guy helps here. I had asked him how to Nim multiply 8x8, and his letter showed how, and also how to multiply 5 by 11.

Here it is. He uses the plus and times signs within circles, and I've substituted + and *. I've also replaced his linear ordering of equal statements with Iverson's convention of placing them one below the other.

> Dear Eugene McDonnell,
>
> Nim-multiplication is tricky, but you can probably catch on by remembering to deal with the exponents in the same way that you deal with numbers in nim-addition, namely split them into powers of 2. Nim multiplication of powers of 2 is defined, in the first instance, only for the 'Fermat powers of 2'

```
(2^2^0)  =  2
(2^2^1)  =  4
(2^2^2)  =  16
(2^2^3)  =  256
(2^2^4)  =  65536
```

each, after the first, being the square of the previous one, but if instead of 'square' you mean 'nim-multiply by itself', than the answer is defined to be

```
(x*x)
(3%2)*x
```

(just if x is a Fermat power of 2).

To deal with other powers of 2, you work at one level higher up, thinking of (2^13), for example, as (2^(8+4+1)) and use the associative, commutative and distributive laws, e.g.,

```
8*8
(2^3)*(2^3)
(2^(2+1))*((2^(2+1))
(2^2)*(2^1)*(2^2)*(2^1)
((2^2)*(2^2))*((2^1)*(2^1))
(4*4)*(2*2)
6*3
(4+2)*(2+1)
(4*2)+(4*1)+(2*2)+(2*1)
8+4+3+2
13
```

To deal with numbers which are not powers of 2 leads to a corresponding extra level of complication. E.g.,

```
5*11
(4+1)*(8+2+1)
(4*8)+(4*2)+(4*1)+(1*8)+(1*2)+(1*1)
(4*4*2)+8+4+8+2+1
(6*2)+7
((4+2)*2)+7
(4*2)+(2*2)+7
8+3+7
12
```

8 is the first power of 2 that is not a Fermat power, and the first place where you run into any difficulty.

Best wishes,

Yours sincerely,

Richard K. Guy,
Faculty Professor of Mathematics
University of Calgary

I end with a complete 16×16 Nim multiplication table:

```
+--+------------------------------------------------+
|  | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|
+--+------------------------------------------------+
| 0| 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|
| 1| 1  0  3  2  5  4  7  6  9  8 11 10 13 12 15 14|
| 2| 2  3  0  1  6  7  4  5 10 11  8  9 14 15 12 13|
| 3| 3  2  1  0  7  6  5  4 11 10  9  8 15 14 13 12|
| 4| 4  5  6  7  0  1  2  3 12 13 14 15  8  9 10 11|
| 5| 5  4  7  6  1  0  3  2 13 12 15 14  9  8 11 10|
| 6| 6  7  4  5  2  3  0  1 14 15 12 13 10 11  8  9|
| 7| 7  6  5  4  3  2  1  0 15 14 13 12 11 10  9  8|
| 8| 8  9 10 11 12 13 14 15  0  1  2  3  4  5  6  7|
| 9| 9  8 11 10 13 12 15 14  1  0  3  2  5  4  7  6|
|10|10 11  8  9 14 15 12 13  2  3  0  1  6  7  4  5|
|11|11 10  9  8 15 14 13 12  3  2  1  0  7  6  5  4|
|12|12 13 14 15  8  9 10 11  4  5  6  7  0  1  2  3|
|13|13 12 15 14  9  8 11 10  5  4  7  6  1  0  3  2|
|14|14 15 12 13 10 11  8  9  6  7  4  5  2  3  0  1|
|15|15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0|
+--+------------------------------------------------+
```

This is identical with Table 10.2 in the Conway & Guy book.

Dyalog.Net: An Introduction and Overview

by John Daintree, Senior Programmer, Dyadic Systems Ltd
email: johnd@dyadic.com, mailing list: dotnet@dyadic.com

Introduction

This is the first of three papers discussing Dyadic Systems' new .Net compatible product, Dyalog.Net. In this paper we introduce Dyalog.Net and give a broad overview of the facilities available. Subsequent papers will focus on particular aspects of the integration with Microsoft. Net.

Dyadic Systems has set up a mailing list to discuss Dyalog.Net related issues. To subscribe to this list, send an email to dotnet@dyadic.com with SUBSCRIBE as the subject.

Setting the Scene

What would *you* do if Bill called?

It was back in 1999 that Dyadic Systems received a phone call from Microsoft. OK, the call wasn't from Bill Gates himself (who, incidentally has an interest in APL), but from Microsoft employees who described themselves as "Software Evangelists". Would Dyadic Systems Ltd be interested in participating with Microsoft in the development of a "Major New Technology"?

Duh! we said "Yes".

Shortly thereafter we received a visit from two Microsoft representatives who had come from Seattle to see us. And yes, their business cards did say "Software Evangelist".

The product Dyalog.Net is the result of a collaboration between Dyadic Systems and Microsoft. Dyalog.Net combines the power of Dyalog APL with the benefits of the Microsoft .Net Framework.

What is Microsoft .Net?

Anyone who has even glanced at a computer related publication over the past 6 to 24 months could not have helped but become aware of Microsoft .Net.

The .Net Framework is, to quote Microsoft,

"...a set of Microsoft software technologies for connecting your world of information, people, systems, and devices. It enables an unprecedented level of software integration ...., discrete, building-block applications that connect to each other—as well as to other, larger applications.... .NET connected software delivers what developers need to create XML Web services and stitch them together. The benefit to individuals is seamless, compelling experiences with information sharing."

From an APL perspective Microsoft .Net provides us with a large number of objects, i.e. instances of classes (or, to use Microsoft .Net parlance, *Types*). These Types are distributed within dynamic link libraries (or *Assemblies*). Types also follow a dot-delimited naming convention, conveniently called *Namespaces.*

So, for example, there is a class called **File** that resides in a namespace called **System.IO**, to give a complete name for the Type of **System.IO.File**. This class allows us to manipulate files in an object-oriented fashion.

The Microsoft .Net Framework, as shipped by Microsoft, consists (amongst other things) of a large number of assemblies and the infrastructure required to host them.

Assemblies are self-describing, that is to say that an Assembly not only contains the binary code to implement all the methods in the assembly but also data that describes every method in the assembly. In addition every parameter to every method in every type in every namespace is described in this data. This extremely valuable source of information is termed *Metadata.*

Metadata

It is tempting to compare metadata with Type Library information, which we may be familiar with from COM. However type library information was essentially optional within the COM environment (although Dyalog APL requires it).

Metadata is an integral part of an assembly. Without metadata the assembly is incomplete and there is no way to access the methods within it (from Dyalog.Net or from any other programming language). Gone are the days of missing type libraries or worse, incorrect or inconsistent type libraries.

Inheritance.

The .Net model is a rich Object-Oriented model. With one single exception every Type derives from (i.e. *inherits* from) a single base-type. This inheritance can be arbitrarily deep, but ultimately every Type derives from a Type called **System.Object**. **System.Object** is the exception to this rule, as it derives from nothing; it is the "Grand-Daddy of them all".

What is Dyalog.Net?

Dyalog.Net is a companion product to Dyalog APL 9.0. Dyalog.Net requires that the Microsoft .Net Framework is installed. In fact Dyadic Systems recommends that the .Net SDK be installed on the machine, as this package includes a number of useful tools, a large amount of documentation and a vast number of useful examples.

Dyalog.Net allows APL programmers to make use of the Types defined with assemblies, including those Types that are part of Microsoft .Net itself, and also new Types written in a number of languages, such as VB.Net, or Microsoft's new .Net specific language, C#, (pronounced "C-Sharp").

Dyalog.Net also allows APL programmers to create their own Types that can subsequently be used by .Net aware applications, which in turn may be written in any .Net aware language.

Why?

Amongst the classes included within the .Net Framework are Types such as **System.Web.UI.Page**, which allows us to write Web Pages without any need to write the TCPIP "plumbing code". **System.Web.WebServices.WebService**, which allows us to write an application that can be called over the Internet by any WebService aware application. There is a namespace called **System.Windows.Forms**, which contains a number of Types that allow us to manipulate the GUI of the host operating system.

In addition there are a large number of "primitive" types such as 64-bit integers, currency types, datetime types and so on, that can be used as part of an APL application.

In other words there are a large number of extremely useful classes that we want to be able to utilize.

How?

Using Existing Types.

Dyalog.Net is fully compatible with Dyalog APL 9.0 and is able to run all existing Dyalog APL code and applications. To preserve this level of backwards compatibility Dyadic Systems has introduced a new system variable, ⎕*USING*, to determine that .Net Types are to be used by the application. ⎕*USING* allows the programmer to specify which .Net classes to use, and in addition, in which assembly these Types are located.

The simplest specification of ⎕*USING* is a single empty character vector.

```
      ⎕using←,⊂''
      ⍴⎕using
1
```

and this can be abbreviated to

```
      ⎕using←''
      ⍴⎕using
1
```

Most of the basic .Net types are defined within an Assembly called mscorlib.dll. Dyalog.Net automatically makes this Assembly available so we do not need to specify an assembly name in ⎕*USING*.

We can now access some of the "primitive" .Net objects

```
      dt1←System.DateTime.New 2002 8 31 7 30 0
      dt1
31/08/2002 07:30:00
```

The **DateTime** Type is a class that allows is to manipulate dates and times as primitive objects.

```
      ⎕nc 'dt1'
9
```

We can see that *dt1* is an object of class 9, i.e. it is a "namespace reference", or more precisely a reference to a .Net object.

The display form of a .Net object is obtained from a method within the object called **ToString**. This is a method that is defined on **System.Object**, and because every object in the .Net framework derives (ultimately) from **System.Object** we should be able to call **ToString** on all objects. Dyalog.Net calls the **ToString** method whenever the display form of a .Net object is required. We could easily type

```
      dt1.ToString ⍬
31/08/2002 07:30:00
```

To get the same result

Many of the methods within the .Net framework exist in a number of forms, where each form may take a different number of parameters, or where the parameters may have different types. This is a concept called *overloading*.

Dyalog.Net automatically matches your parameters with each of the possible sets for the method and picks the version of the method that matches what you have supplied.

In this instance we want to call the *overload* of the **ToString** function that takes zero arguments, so we provide an empty parameter list, specified by ⍬.

The *New* method, that we called to create `dt1`, is known as the *constructor* for the Type. That is, it can be used to create an initialised instance of the Type. The constructor for the **DateTime** class is overloaded, and in the example above we used the overload that took 6 numeric parameters. Alternatively we could have called the overload that takes only three arguments.

```
      dt2←System.DateTime.New 2002 8 31
      dt2
31/08/2002 00:00:00
```

We can see that in this case we have specified the date part of the **DateTime**, and defaulted the time section.

We can determine the exact type of a .Net object by calling its GetType member. Again, this member is defined on System.Object and so is available on all objects. GetType returns an instance of the System.Type class. The display form is again obtained from ToString, and so is easily readable.

```
      type←dt2.GetType
      type
System.DateTime
```

The **ToString** member of the **DateTime** object has a number of overloads, some of which take arguments, and one of which does not. In APL we have no mechanism to have an optional right argument to a function so **ToString** appears as a monadic function, and we use ⍬ to indicate an empty parameter list. **GetType** only has a single overload, and this take no arguments, so **GetType** appears as a niladic function in the object.

We can use the result of **GetType** to find out more about the instance, for example, the list of Constructors that are available to us:

```
       ↑⍕¨dt2.GetType.GetConstructors ⍬
 Void .ctor(Int64)
 Void .ctor(Int32, Int32, Int32)
 Void .ctor(Int32, Int32, Int32, System.Globalization.Calendar)
 Void .ctor(Int32, Int32, Int32, Int32, Int32, Int32)
 Void .ctor(Int32, Int32, Int32, Int32, Int32, Int32,
      System.Globilization.Calendar)
 Void .ctor(Int32, Int32, Int32, Int32, Int32, Int32, Int32)
 Void .ctor(Int32, Int32, Int32, Int32, Int32, Int32, Int32,
      System.Globalization.Calendar)
```

These are the methods that the function *New* calls internally. The result of **GetConstructors** is actually an array of **MethodInfo** objects. The display form of a **MethodInfo** is a C language-like description of the method. You will notice that format, ⍕, calls the **ToString** member of the object.

The documentation for the **DateTime** class describes each of these overloads in detail.

As mentioned previously we can manipulate these **DateTime** objects as if they were primitive objects in APL, for example

```
       diff←dt1-dt2
       ⎕nc 'diff'
 9
       diff.GetType
 System.TimeSpan
       diff
 07:30:00
       diff.TotalSeconds
 27000
```

Subtraction of **DateTime** objects results in an instance of a **TimeSpan** object. Dyalog.Net calls the **op_Subtraction** method defined on the **DateTime** Type. This method does the actual calculation and returns the result, which again is a namespace reference. In the same way Dyalog.Net can perform addition, multiplication, comparison, and other common operations on .Net objects.

Scalar extension works in exactly the way we would expect

```
      dt2 dt1-dt2
00:00:00  07:30:00
      (dt2 dt1-dt2).TotalSeconds
0 27000
```

In the examples so far, all the methods we have called have been methods on instances of the Type. These are called *instance* methods. However, there are a number of methods that we can call directly on the class. These are called *static* methods, for example

```
      System.DateTime.IsLeapYear ¨1600 1601
1 0
```

We can also have static properties on .Net Types

```
      now←System.DateTime.Now
      now
27/05/2002 10:21:32
```

The static property **Now** (unsurprisingly) provides the current date and time.

.Net objects can generate errors. Within the .Net framework errors are handled in a very APL like way. When an error is raised the calling method can trap the error and handle it accordingly. If the error is not caught the application is interrupted and may either terminate or enter a debugger.

Within the .Net framework, errors are called *Exceptions.* Exceptions are raised (i.e. *thrown*) and trapped (i.e. *caught*). In Dyalog.Net, when a .Net method throws an exception an *EXCEPTION* error is signalled. When an *EXCEPTION* is signalled the system variable ⎕*EXCEPTION* is set to the value of the thrown exception. This is itself an instance of a .Net object, specifically something that derives from **System.Exception**. The exception object has members such as **Message**, which return a text-description of the error, and **StackTrace**, which returns the execution stack at the point the error was thrown.

The **DateTime** class has a static method called **Parse**, which takes a string argument and attempts to convert the string to an instance of **DateTime**.

```
      dt1←System.DateTime.New 2002 8 31
      str←dt1.ToString ⍬
      dt2←System.DateTime.Parse str
      dt2=dt1
1
```

If we call **Parse** with an invalid string an Exception is raised

```
      System.DateTime.Parse '27/05/2002a'
EXCEPTION
```

and we can interogate the exception

```
      ⎕exception.Message
The string was not recognized as a valid DateTime.  There is an
 unknown word starting at index 10.
      ⎕exception.GetType
System.FormatException
```

`EXCEPTION` has the value 90 and can be trapped in the usual way.

The use of System as a prefix to all the references to **DateTime** may be something of a nuisance. The syntax of `⎕USING` allows us to omit this part of the name

```
      ⎕using←'System'
      DateTime.Now
27/05/2002 10:29:30
```

The value of each element of `⎕USING`, is prepended to the classname until a match is found.

We can use multiple elements in `⎕USING` to access components from multiple namespaces in multiple assemblies, for example to get the SOAP representation of an array

```
      ⎕using←,⊂'System' 'System.IO'
      ⎕using,←⊂'System.Runtime.Serialization.Formatters.Soap,
                System.Runtime.Serialization.Formatters.Soap.dll'
```

In this example, the third element of `⎕USING` includes the filename of the assembly. This is separated from the namespace specification by a comma. Dyalog.Net loads this assembly into memory, thus making the Types defined in it available.

We can then write fairly trivial serialization code to get the SOAP for the array. (With thanks to Stefano Lanzavecchia who originally posted similar code to dotnet@dyadic.com)

```
      fmt←SoapFormatter.New ⍬
      ms←MemoryStream.New ⍬
      a←((2 3 ⍴⍳6) (2 2⍴'hello' 'world') (.1+⍳3))
      fmt.Serialize ms a
      soap←82 ⎕dr ms.ToArray
```

```
    soap

<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
stance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-E
NC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="ht
tp://schemas.xmlsoap.org/soap/envelope/" xmlns:clr="http://schema
s.microsoft.com/soap/encoding/clr/1.0" SOAP-ENV:encodingStyle="ht
tp://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:anyType[3]">
<item href="#ref-2"/>
<item href="#ref-3"/>
<item href="#ref-4"/>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="ref-2" SOAP-ENC:arrayType="xsd:short[2,3]">
<item>1</item>
<item>2</item>
<item>3</item>
<item>4</item>
<item>5</item>
<item>6</item>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="ref-3" SOAP-ENC:arrayType="xsd:anyType[2,2]"
<item id="ref-5" xsi:type="SOAP-ENC:string">hello</item>
<item id="ref-6" xsi:type="SOAP-ENC:string">world</item>
<item id="ref-7" xsi:type="SOAP-ENC:string">hello</item>
<item id="ref-8" xsi:type="SOAP-ENC:string">world</item>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="ref-4" SOAP-ENC:arrayType="xsd:double[3]">
<item>1.1</item>
<item>2.1</item>
<item>3.1</item>
</SOAP-ENC:Array>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Creating new Types

In addition to being able to make use of existing Types, Dyalog.Net allows us to define new Types (that derive from any appropriate existing Type). This can be achieved in a number of ways. In this paper we will consider the traditional mechanism, `⎕WC`, that will be familiar to most users of Dyalog APL.

We want to create a new .Net Type, so we use `⎕WC` to create a new namespace. We can (optionally) provide the name of the Type from which we wish to derive. The name of the base type is resolved using the current value of `⎕USING`, so the following statements are equivalent:

```
⎕using←'System'⋄  'Demo'⎕WC'NetType' 'Object'
⎕using←0⍴⎕using   'Demo'⎕WC'NetType'
⎕using←''      ⋄  'Demo'⎕WC'NetType' 'System.Object'
```

Once we have created the namespace we move into that space

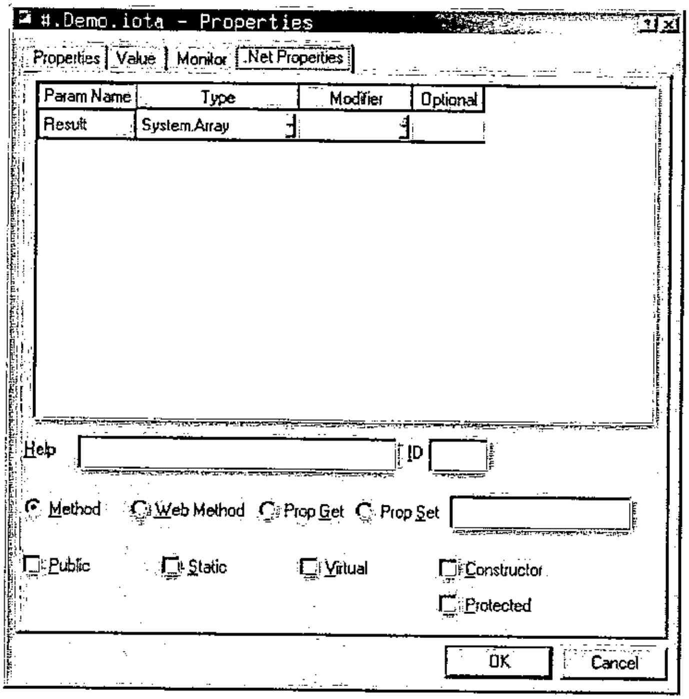
```
      )cs Demo
#.Demo
```

We now define an APL function in any of the usual ways

```
      ⎕fx 'r←iota n' 'r←⍳n'
      )fns
iota
```

As already stated, a .Net assembly includes metadata that describes the parameters to each of the methods in the assembly. From the Dyalog.Net development environment we use a dialog box to provide this information.

We can right click on the name of a function, and select the "Properties" menu item. This presents us with the properties dialog box for the function.

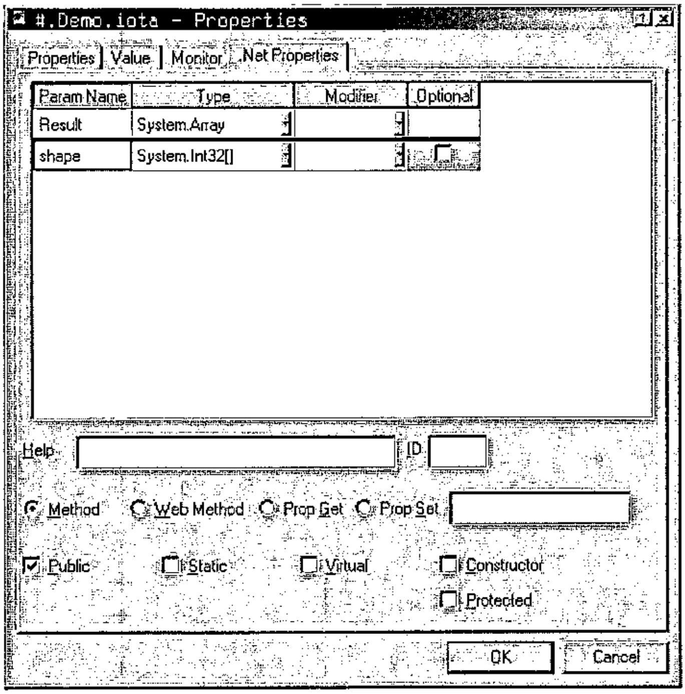


Param Name	Type	Modifier	Optional
Result	System.Array		

[✓ Method] Web Method Prop Get Prop Set

We can modify this box to provide the correct information about the method that we wish to export.

From an APL perspective, the method iota can take a rank 1 array of integers as its parameter. The result will be an **Array** of corresponding rank. The following dialog box contains the appropriate definitions.



#.Demo.iota - Properties

Param Name	Type	Modifier	Optional
Result	System.Array		
shape	System.Int32[]		☐

Help: ID:

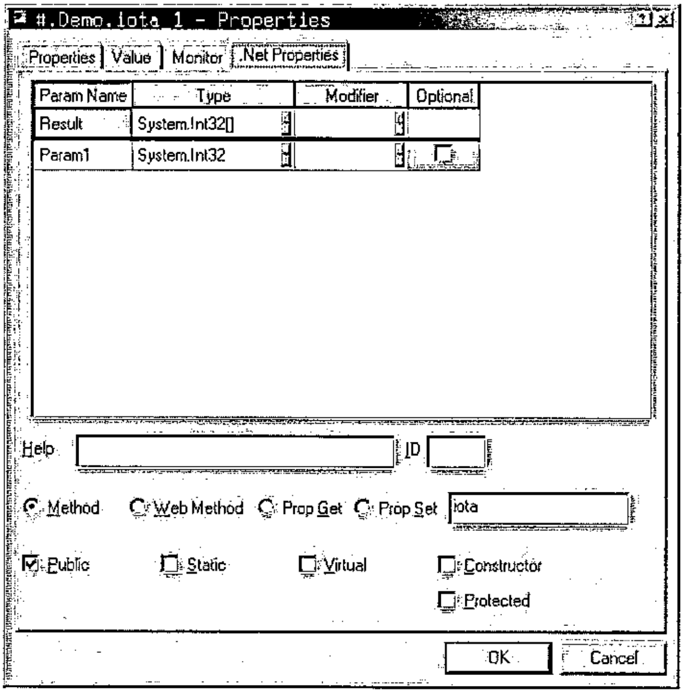
[✓ Method] Web Method Prop Get Prop Set

[✓ Public] ☐ Static ☐ Virtual ☐ Constructor ☐ Protected

In this illustration "System.Int32[]" is the .Net specification of a rank 1 array of 32-bit integers. Similarly "System.Array" is the .Net specification of an **Array** of unknown type and unknown rank.

This is the closest match we can get to the APL definition of the iota primitive, however other languages with a less elegant mechanism for creating arrays would benefit from a simpler declaration of the function. For this reason it is good practice to provide overloads for the common cases in addition to the generic solution.

Here is the properties box for a function called iota_1 that provides the metadata for the rank 1 variant of iota. Here the result is a rank 1 array of integers and the parameter is a scalar 32-bit integer. Note that we can specify the "external" name for iota_1, in this case "iota", thus specifying that we are providing an overload for that function.


#.Demo.iota_1 - Properties

Properties | Value | Monitor | .Net Properties

Param Name	Type	Modifier	Optional
Result	System.Int32[]		
Param1	System.Int32		☐

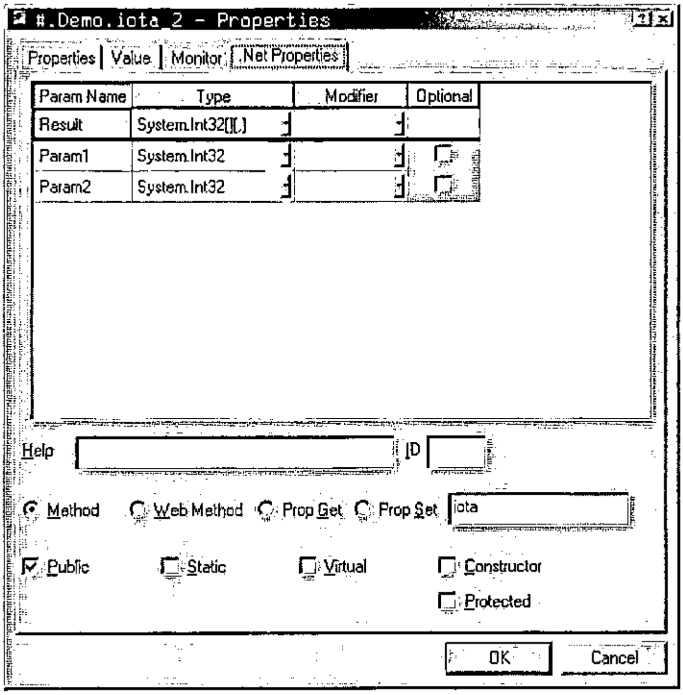
Help: ID:

[✓ Method] Web Method Prop Get Prop Set iota

[✓ Public] Static Virtual Constructor Protected

OK Cancel

Similarly here is the properties box for a function called iota_2 that provides the metadata for the rank 2 variant of iota.



#.Demo.iota_2 - Properties

Properties | Value | Monitor | .Net Properties

Param Name	Type	Modifier	Optional
Result	System.Int32[][,]		
Param1	System.Int32		☐
Param2	System.Int32		☐

Help: ID:

[✓ Method] Web Method Prop Get Prop Set: iota

☑ Public ☐ Static ☐ Virtual ☐ Constructor ☐ Protected

OK Cancel

Note that here, because we need to return a nested array, the result type is declared as a 2 dimensional array of Int32[].

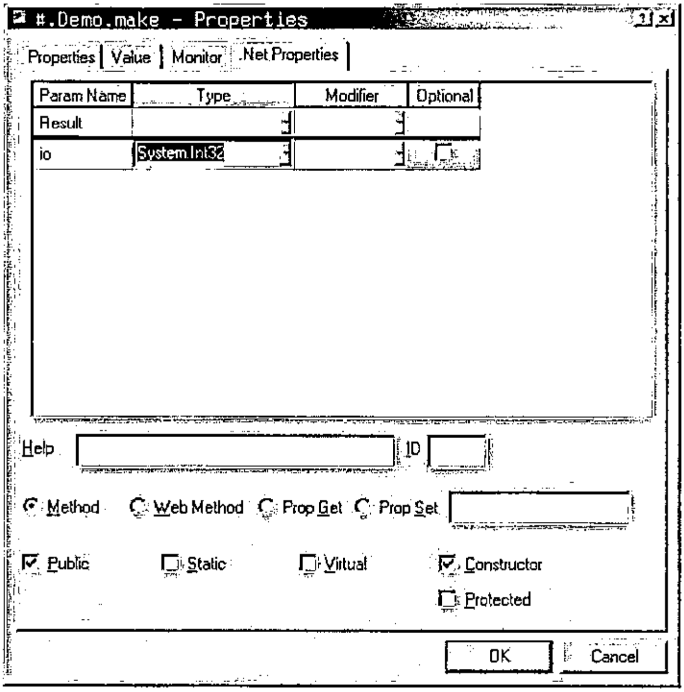
Finally we may want to specify the index origin that should be used when iota is executed. We could pass an extra argument to the iota method each time it is called, but a better solution would be to define a constructor method for the Demo class. This constructor could have a single argument that is the value of ⎕IO to be used during the iota calculations.

At runtime, each instance of our Demo class is its own instance of the Demo namespace. As ⎕IO has namespace scope, we can be sure that each instance of Demo will execute independently of any other.

Here is the body ~~and~~ of our constructor method.

```
     ∇ make io
[1]
[2]    :Trap 0
[3]        ⎕IO←io
[4]    :Else
[5]        (Exception.New'io must be 0 or 1')⎕SIGNAL 90
[6]    :End
     ∇
```

Note that we trap any error from the assignment to ⎕IO. If there is an error we use ⎕SIGNAL to raise a .Net Exception, with an appropriate message.



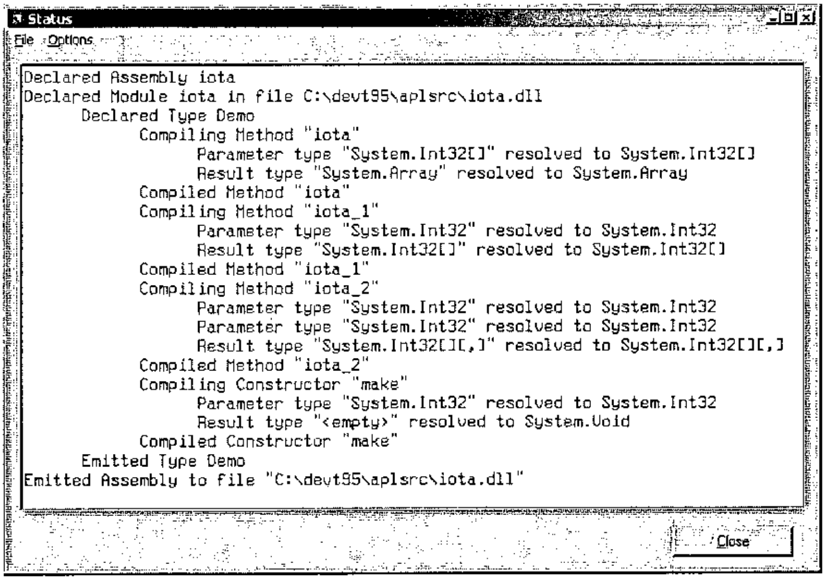
Param Name	Type	Modifier	Optional
Result			
io	System.Int32		☐

[✓ Method] Web Method Prop Get Prop Set

[✓ Public] Static Virtual [✓ Constructor] Protected

Note that we have checked the "Constructor" check box, and that the method does not return a result (the Type field of "Result" is empty). When this method is called we will be inside a newly created instance of the Demo namespace, so the assignment to ⎕IO will only affect this instance of Demo.

Once we have defined all the methods in our Type we can package our workspace as a .Net assembly. This can be achieved with the "Make Assembly" menuitem on the session menu. When we select this item we can choose the name and location of the desired assembly, and Dyalog.Net builds the assembly. The status window is used to provide information about the "compilation" process

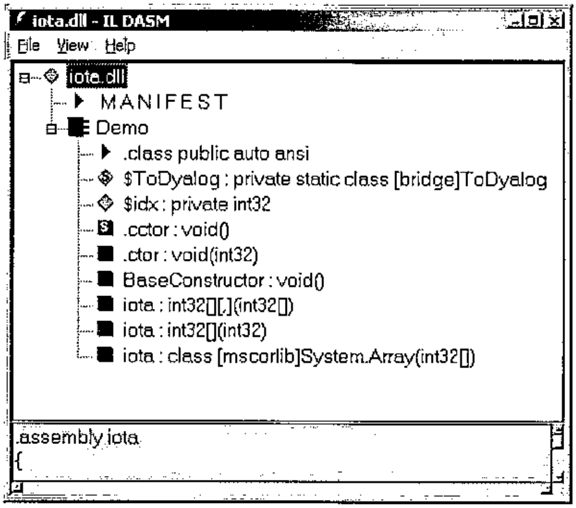


```
Declared Assembly iota
Declared Module iota in file C:\devt95\aplsrc\iota.dll
      Declared Type Demo
            Compiling Method "iota"
                  Parameter type "System.Int32[]" resolved to System.Int32[]
                  Result type "System.Array" resolved to System.Array
            Compiled Method "iota"
            Compiling Method "iota_1"
                  Parameter type "System.Int32" resolved to System.Int32
                  Result type "System.Int32[]" resolved to System.Int32[]
            Compiled Method "iota_1"
            Compiling Method "iota_2"
                  Parameter type "System.Int32" resolved to System.Int32
                  Parameter type "System.Int32" resolved to System.Int32
                  Result type "System.Int32[][,]" resolved to System.Int32[][,]
            Compiled Method "iota_2"
            Compiling Constructor "make"
                  Parameter type "System.Int32" resolved to System.Int32
                  Result type "<empty>" resolved to System.Void
            Compiled Constructor "make"
      Emitted Type Demo
Emitted Assembly to file "C:\devt95\aplsrc\iota.dll"
```

You will notice the word "compiled" in the Dyalog.Net status window. It is important to note that the APL code itself is not compiled. The process compiles "stub" functions that make calls into the Dyalog.Net interpreter. The word "compile" is used to be consistent with other languages and .Net tools.

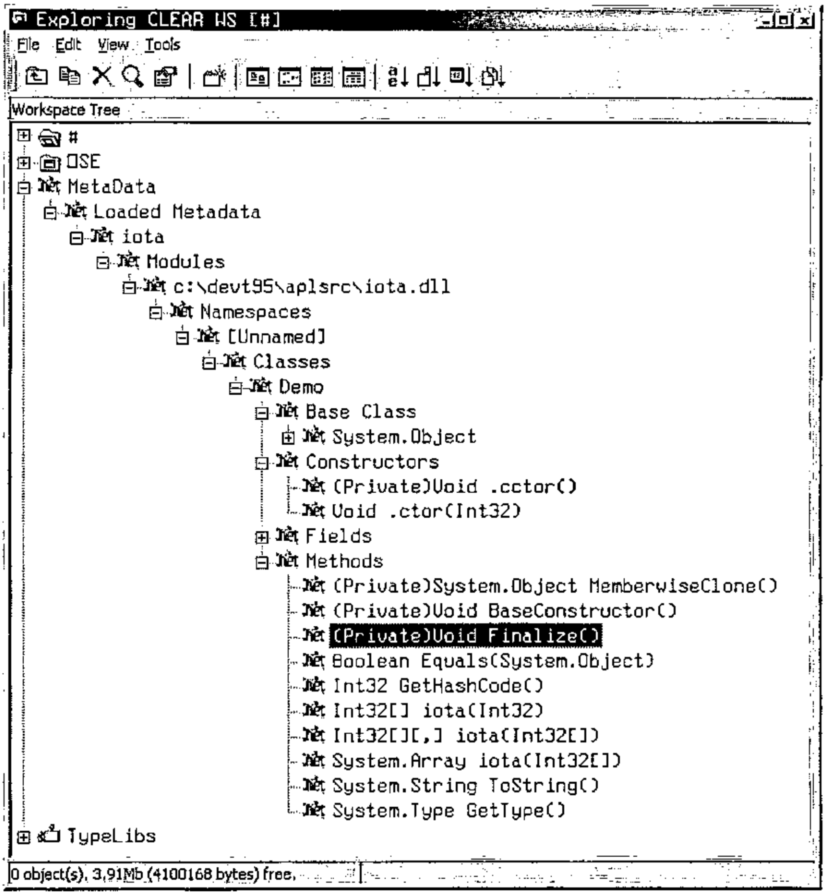
Also, the workspace is not saved at this point, if we want to save our source code, it is necessary to use *)SAVE* to save the workspace. The Assembly that we have created includes the active workspace so we do not need to ship our workspace with the generated dynamic link library.

We can use Microsoft's ILDASM tool, to investigate the contents of our newly created assembly, and we can see the several overloads of the iota function, and also our definition of the constructor, called ".ctor".



```
iota.dll
    MANIFEST
    Demo
        .class public auto ansi
        $ToDyalog : private static class [bridge]ToDyalog
        $idx : private int32
        .cctor : void()
        .ctor : void(int32)
        BaseConstructor : void()
        iota : int32[][,](int32[])
        iota : int32[](int32)
        iota : class [mscorlib]System.Array(int32[])
.assembly iota
{
```

Of course, Dyalog.Net has a built-in Metadata viewer. This is part of the Workspace Explorer.



```
# 
⎕SE
MetaData
  Loaded Metadata
    iota
      Modules
        c:\devt95\aplsrc\iota.dll
          Namespaces
            [Unnamed]
              Classes
                Demo
                  Base Class
                    System.Object
                  Constructors
                    (Private)Void .cctor()
                    Void .ctor(Int32)
                  Fields
                  Methods
                    (Private)System.Object MemberwiseClone()
                    (Private)Void BaseConstructor()
                    (Private)Void Finalize()
                    Boolean Equals(System.Object)
                    Int32 GetHashCode()
                    Int32[] iota(Int32)
                    Int32[][,] iota(Int32[])
                    System.Array iota(Int32[])
                    System.String ToString()
                    System.Type GetType()
TypeLibs
0 object(s), 3,91Mb (4100168 bytes) free,
```

This is the C# code that uses instances of our Demo class to perform the iota calculations. We can clearly see the advantages of using the "simple" overloads of iota.

```
using System;
class UseIota
        {
        static void Main()
                {
                Demo demo = new Demo(0);
                int[] shape={1,2,3};

                Console.WriteLine(demo.iota(10));

                Console.WriteLine(demo.iota(10,10));
```

```
            demo = new Demo(1);

            Console.WriteLine(demo.iota(shape));
            }
      };
```

Future Papers

APLScript

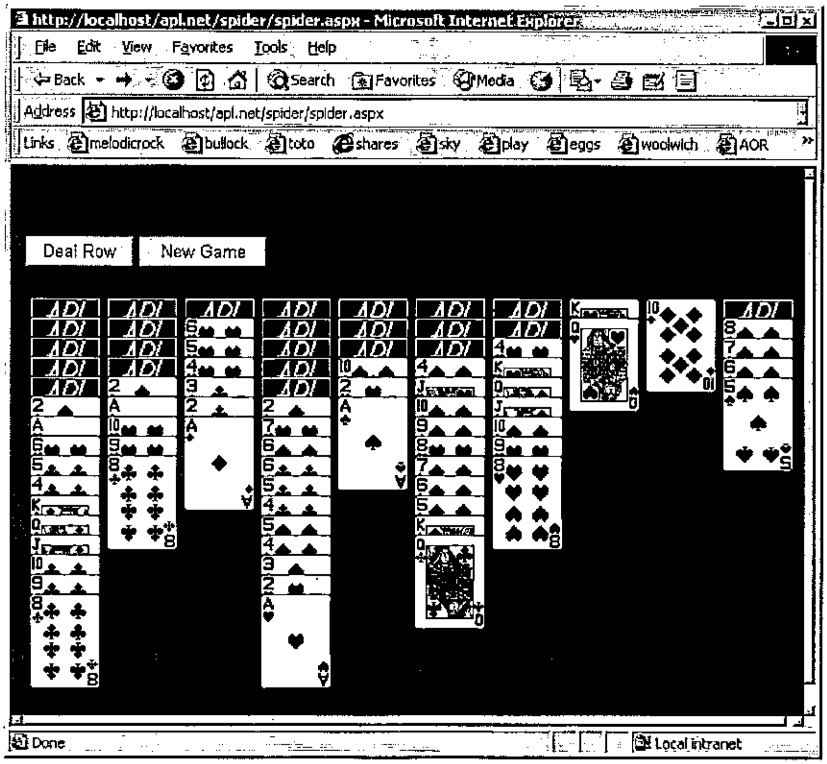
Using the development environment to create .Net Assemblies is a convenient mechanism for those of us who are familiar and comfortable with the IDE. On many occasions it is more convenient to define our source code using a more traditional mechanism. This is where APLScript comes in. The following example lists the same Demo class, this time, written in APLScript. This is saved in a single UNICODE text file, and we use the APLScript compiler to generate the assembly. APLScript will be discussed in detail in a following paper.

```
  :Class Demo

      ∇make io
      :Access Constructor
      :ParameterList Int32
      :Trap 0
           ⎕IO←io
      :Else
           (Exception.New'io must be 0 or 1')⎕SIGNAL 90
      :End
      ∇

      ∇r←iota n
      :Access Public
      :ParameterList Int32[]
      :Returns Array
      r←⍳n
      ∇

      ∇r←iota_1 n
      :Access Public
      :ParameterList Int32
      :Returns Int32[]
      :Implements Method iota
      r←iota n
      ∇

      ∇r←iota_2 n
      :Access Public
      :ParameterList Int32[]
      :Returns Int32[][,]
      :Implements Method iota
      r←iota n
```

```
        ∇

  :EndClass
```

Web Pages and Web Services

We will see how we can use the .Net framework and APLScript to create web pages such as this one. The multi threading aspects of providing web pages will be discussed.


http://localhost/apl.net/spider/spider.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/apl.net/spider/spider.aspx
Links melodicrock bullock toto shares sky play eggs woolwich AOR
Deal Row
New Game
Done
Local intranet

The .Net GUI: System.Windows.Forms

The System.Windows.Forms namespace contains a rich and varied set of classes that can be used to create GUI applications. These classes are available to us either as an alternative to, or in addition to, the native Dyalog GUI.

Bitmaps in Dyalog APL/W

by David Crossley (crossley@au-village.com)

It used to work...

Dyalog APL/W provides a workspace called BMED that contains two basic but quite useful utilities. The first is also called BMED to create or edit the the Bits to define a Bitmap. The second is called BTNED to create or edit the Bits needed to define the three Bitmaps identified in the BtnPix Property of a Push Button Object, each providing the image for its different states (Out, In and Inactive). In both cases, the other important ingredient, the Colour Map (CMap) assumes the standard 4-bit (16 colour) Windows default.

[For those unfamiliar with Bitmaps, the CMap Property is a 3-column matrix whose rows each define the Red-Green-Blue intensity levels that when combined form a particular colour, the intensity level being measured on a scale from 0 (colour absent) to 255 (full saturation). Thus pure red is 255 0 0, black is 0 0 0, and white is 255 255 255. The Bits Property is a matrix of the size of the Bitmap in pixels where each element is a 0-origin index into a row of CMap.]

The dialogs for the BMED utilities present an editing area wherein a Bitmap is represented by a grid in which each pixel is blown up to a size of, say, 10 by 10 display pixels, or whatever the user prefers. These squares may then be rendered from a palette of colours using mouse clicks , the rather limited set of the 16 standard Windows colours in this instance. In addition, a number of tools are provided including some to draw graphical objects (lines, rectangles and ellipses). An ellipse would look nice, I thought. Just click on the Ellipse tool, click on a starting "pixel" and then drag the bounding rectangle. The result? No ellipse, nor any change at all to my work or art! Yet I felt sure that this used to work.

What went wrong?

The technique takes advantage of the Ellipse Object to actualy draw the ellipse, illustrated as follows:

```
'ff'⎕WC'Form' '' (10 10)(100 100)('Coord' 'Pixel')('Bitmap' 'ff.bm')
```

This line is not part of the code. It just provides a visual display of what is happening if you wish to try this for yourself. These are the important lines:

```
'ff.bm'⎕WC'Bitmap' ('Bits' (24 24⍴0))
'ff.bm.'⎕WC'Ellipse'(7 4)(10 16)('FCol' 255 255 255)
bits←0≠'ff.bm'⎕WG'Bits' 'CMap'
```

First create a Bitmap Object of the actual required size for the finished product but with all elements set to `0`, the index of black in the default Colour Map, though the actual colour is not important. Then draw an Ellipse Object into the Bitmap from the desired start point (7,4) and size (10,16) that define the bounding rectangle for the ellipse. Specify a colour for the resulting ellipse; `255 255 255` is white though again this does not matter as long as it differs from the background. Note that it is not necessary to give the Ellipse Object a name thereby saving Windows resources, a useful property of drawing into a Bitmap. Any number of graphical objects may be added to the image. In this example, the Form will display an ellipse with a white outline against a black background.

To obtain the outline of the ellipse, simply get the Bits Property and "not-equal" to `0`. This result may then be used to map the 0-origin index of the actual colour we wish to use, 9 for Red, say, into the working Bitmaps's Bits variable, _BITS_ say, and then into the editing display. For example:

```
((,bits≠0/,BITS)←9
```

Unfortunately, this did not work. Investigation revealed...

```
⍴¨'ff.bm'⎕WG'Bits' 'CMap'
 0 0  0 3
```

And the reason?

Like most people these days, the colour display resolution for my screen is set to True Colour (24-bit colour). When reduced to 256 colours (8-bit colour), the method indeed worked. So my memory was not at fault!

It turns out that Dyalog APL does not fully support Bitmaps with 24-bit colour. In the above, when the initial Bitmap is defined with Bits set to all zeroes, a Colour Map (CMap) with just 2 colours is initialised, actually the first two from the standard Windows 4-bit colour table. Both Bits and CMap may be successfully retrieved. However, as soon as a child object is drawn into the Bitmap image, the colour representation converts, it would seem, according to the current colour resolution of the display. Whether this is done by Dyalog or Windows, I cannot say. With a colour resolution of 8-bits, the Colour Map converts to 256 colours. For True Colour, a 24-bit colour scheme is adopted. With an 8-bit setting or lower, both Bits and CMap may be retrieved from the Bitmap Object. However, the way in which colour is handled in a Bitmap resource alters for a 24-bit scheme, as will be discussed further.

Dyalog APL can display a Bitmap with 24-bit colour, as confirmed by the Form display above. It can also read and display a Bitmap with 24-bit colour from a .BMP file. However, retrieving Bits and CMap using `⎕WG` gives results with shape `0 0` and `0 3` respectively. Not very useful.

More about Bitmaps

The full specification for a Bitmap supports 1-bit (2 colours), 4-bit (16 colours), 8-bit (256 colours) and 24-bit colour (about 16.5 million colours, also known as True Colour). Up to 8-bit colour, the actual colours are stored in a separate Colour Map and the values of the Bitmap Bits are 0-origin indices into this map. For 24-bit colour, there is no Colour Map. Instead, each element of Bits is an encoding of the Red-Green-Blue (RGB) intensity values on a scale from 0 (colour absent) to 255 (full saturation).

A Bitmap .BMP file [1] has the following structure which is more formally documented in the leading comments for function `BM_Read` in the Appendix:

- BITMAPFILEHEADER

  File header information giving the file identification ('BM'), total file size, and offset to the Bits data.

- BITMAPINFOHEADER

  Information about the Bitmap, including its width and height, colour scheme (1, 4, 8 or 24 bits), and actual number of colours used in the Colour Map (this defaults to 2, 16 or 256 if not specified).

- RGBQUAD

  The Colour Map specifying the RGB colour intensities. This is absent for 24-bit colour. Each colour is represented in 4 bytes in the order Blue- Green-Red with the 4th byte set to 0.

- Bitmap Bits

  The Bits data for each row of pixels starting at the *last* row with each row padded out to a 4-byte boundary if necessary. Each element takes 1, 4, 8 or 24 bits depending on the colour scheme in use. In the first three cases, the values represent 0-origin indices into the Colour Map; in the latter case, the 24-bits represent 8-bits per primary colour in the order Blue-Green-Red.

A Bitmap resource is represented by Windows using the same structure but omitting the BITMAPFILEHEADER structure.

Using this knowledge, I devised a method to overcome the True Colour problem. The function BM_Read will read a given .BMP file and extract the Bits and CMap information. It returns a 2-item vector whose first element is a boolean flag, 1 for success or 0 for failure, and the second contains the Bits and CMap results which have shape `(0 0)` and `(0 3)` respectively in the event of failure, eg. if the file does not exist or it is not a Bitmap file.

If the file contains a 24-bit colour Bitmap, it is reduced if possible to use the smallest Colour Map attainable depending on the number of unique colours actually found. If there are more than 256 colours, Bits contains RGB values encoded to base `256 256 256` in the order Red-Green-Blue which is consistent with the CMap order. In this case, CMap has shape (`0 3`) which should be used to check the meaning of elements of Bits. Unfortunately, although as noted above the Bitmap can be defined directly from the .BMP file, Dyalog APL will not accept Bits with RGB values along with an empty CMap to define a Bitmap, Cursor or Icon (a *DOMAIN ERROR* is reported) , perhaps something that might be rectified by the vendors? However, we now have access to and can utilise the Bits information in whichever form it is returned.

A work-around solution

Restating the example, we can now write a code fragment as follows:

```
'ff.bm'⎕WC'Bitmap'('Bits'(24 24⍴0))
'ff.bm.'⎕WC'Ellipse' (7 4)(10 16)('FCol' 255 255 255)
:If 0∊⍴bits←'pn.bm'⎕WG'Bits'
'ff.bm'⎕WS'File' 'tmp.bmp' ⋄ 1 ⎕NQ 'ff.bm' 'FileWrite'
bits cmap←2⊃BM_Read'tmp.bmp'
erase 'tmp.bmp'
:If 0<⊃⍴cmap
    bits←bits≠¯1+(↓cmap)⍳⊂0 0 0
:EndIf
:EndIf
bits←bits≠0
```

After creating the Bitmap with an ellipse, a test determines whether the Bitmap is usable, which is the case only if the screen colour resolution is 8-bit or less. If not, we create a .BMP file from the Bitmap. Examination of this file would verify that it has no Colour Map and that the Bits are RGB values, even though there are only 2 colours actually used. Reading the file with function BM_Read, the Bitmap is converted to use a 2-bit Colour Map. Checking the shape of CMap is redundant here, though required in a more general application.

Just to confirm...

```
      ⍴bits
24 24
      ⍴cmap
2 3
```

I expect there is a slicker way of doing this using a Windows API call through ⎕NA though the principle remains the same and this method seems quick enough on my pre-historic 266MHz Pentium.

A concluding thought

The same methodology could be used in reverse. Supposing you wished to create a Bitmap Object with 24-bit colour from RGB-encoded Bits that you have perhaps generated. Dyalog APL will not allow you to simply define the Bitmap Object using ⎕WC. However, one could write a complementary function to *BM_Read* , *BM_Write* say, using similar logic that would create a Bitmap .BMP file. Dyalog APL can then generate the required Bitmap Object from this file using ⎕WC.

Reference

[1] Charles Petzold, *Programming Windows 3.1*, Microsoft Press 1992, p607-609

Index to Advertisers

APL Borealis	28
Dyadic Systems Ltd	2
Fred Honea	8
MicroAPL Ltd	48
Vector Back Numbers	7

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.

Subscribing to Vector

Your Vector subscription includes membership of the British APL Association, which is open to anyone interested in APL or related languages. The membership year runs from 1st May to 30th April. 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: ____________________

UK private membership	£20
Overseas private membership	£22
Airmail supplement (not needed for Europe)	£4
UK Corporate membership	£100
Corporate membership overseas	£135
Sustaining membership	£500
Non-voting UK member (student/OAP/unemployed)	£10

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

Signature: ____________________

*Data Protection Act:
The information supplied may be stored on computer and processed in accordance with the registration of the British Computer Society.*

Send the completed form to:
BAA, c/o Rowena Small, 12 Cambridge Road, Waterbeach, CAMBRIDGE CB5 9NJ, UK
Fax: +44 (0) 1653 697719

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.

2001/2002 Committee

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), Anthony & Sylvia Camacho (0117-973-0036), Ray Cannon (01252-874697), Stephen Taylor (077-1340-0852) Bob Hoekstra (01483-771028), Jon Sandles (01904-612882), Marc Griffiths

Typeset by APL-385 with MS Word for Windows

Printed in England by Short-Run Press Ltd, Exeter

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 [illegible] support of the following Association Sustaining Members. In many [illegible] also provide manpower and [illegible] assistance to the Association.

Causeway Graphical Systems Ltd
The Maltings, Castlegate
MALTON, North Yorks YO17 [illegible]
Tel: 01653-696760
Fax: 01653-697719
Email: sales@causeway.co.uk
Web: www.causeway.co.uk

Compass Ltd
Compass House
60 Priestley Road
GUILDFORD, Surrey [illegible]
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

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel: +31 347 342 [illegible]

Insight Systems ApS
Nordre Strandvej 119G
DK-3150 Hellebæk
Denmark
Tel: +45 70 26 13 26
Fax: +45 70 26 13 25
Email: info@insight.dk
Web: www.insight.dk

HMW Computing
Hamilton House,
1 Temple Avenue,
LONDON EC4Y [illegible]
Tel: 0870-[illegible]
Email: HMW@[illegible]

MicroAPL Ltd
The Roller Mill, Mill Lane
Uckfield
E.Sussex TN22 5AA
Tel: 01825-768050
Fax: 01825-749472
Email: MicroAPL@[illegible]
Web: www.microapl.co.uk

Soliton Associates [illegible]
[illegible]
[illegible]

APL2000 Inc
One Research Court
Suite [illegible]
[illegible] MD 20850 USA
[illegible]
[illegible]