VECTOR

APL2002 at Madrid ...

- Cannon & Smith on Pocket APL 46
- Madrid Conference Reports 60
- Coxe on Fractals 65
- Eke thumps the Gui Quad 76
- Taylor on RTF generation 95
- Daintree on APLScript and .NET 114

BCS

The Journal of the British APL Association

ISSN 0955-1433
www.vector.org.uk
Vol.19 No.2 October 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

		Page
Editorial	Stefano Lanzavecchia	3
APL NEWS		
Quick Reference Diary		5
APL2003 Information and Call for Papers		6
British APL Association News		
News from Sustaining Members		6
APL Product Guide	Gill Smith	12
The Education Vector Zark Newsletter Extracts	edited by Jon Sandles	26
Crossword		35
J-ottings 34: Greed – Patterns for the Imminent Collapse of Western Capitalism	Norman Thomson	38
REVIEWS AND CONFERENCE REPORT		
Dyalog APL CE (beta) – First Impressions	Ray Cannon	46
If You Go Down to the Woods Today ... First Experiences with PocketAPL	Adrian Smith	52
APL2002 Conference Reports	Camacho, Clark	60
GENERAL ARTICLES		
Inner Product Fractals from Fuzzy Logics	Angela Coxe	65
A Gui-quad for Graphical User Interaction	van Batenburg et al	76
TECHNICAL SECTION		
Technical Correspondence		90
Perfect Printing	Stephen Taylor	95
A Girl's Best Friend	Phil Chastney	109
Dyalog.Net: APLScript and Things Textual	John Daintree	114
Index to Advertisers		143

Editorial

by Stefano Lanzavecchia

> "What is it?" asked Arthur
> "*The Hitchhiker's Guide to the Galaxy*. It's a sort of electronic book. It tells you everything you need to know about anything. That's its job."
> Arthur turned it over nervously in his hands.
> "I like the cover," he said. " 'Don't Panic.' It's the first helpful or intelligible thing anybody's said to me all day."
> Douglas Adams – *The Hitchhiker's Guide to the Galaxy*

The kind and enveloping warmth that greeted me and kept me company while in Madrid for the APL2002 conference was not enough to hide the fact that I felt something wrong. It was not the venue, although something could be said about the lack of a place where attendees could sit and talk together; it was not the open and pleasant atmosphere of the European capital, despite its distressingly high rate of thefts, attempted and succeeded; it was not the Spanish language that, it's no secret, I am not very fond of, one of those hard to explain, irrational dislikes. I felt as if there was something missing, something that made even the purely social aspects of the gathering somewhat duller than expected.

I am biased but I must admit once more that the highlight of the conference was Dyadic's presentation: the not so new anymore but still exciting Dyalog.NET, on one side, and on the other, project X, finally revealed to be a port to the PocketPC platform of the full-featured interpreter. Now that we are used to high-resolution screens it is unclear to me how people will be able to exploit the possibilities of the ultimate portable device with its cranked screen estate, but it was not so long ago that IBM's CGA ruled in the PC world and there are very bright programmers in the community of array-oriented programming languages.

It would be unfair to forget APL2000's blazing introduction to Web Services and not to congratulate MicroAPL for their reborn APLX, or IBM for their steady introduction of new features in APL2 and their new pricing policy for educational purposes. It would probably be fair to forget Soliton's pathetic presentation on a feature just introduced in their Sharp APL, a very long hour and a half spent describing with a detail not even a reference manual gets into, the so-called control-structures. Still I don't feel like forgiving the speaker simply because he warned the audience before he started that it was going to be boring. Let's face it: even if some purists still haven't accepted them, more for academic reasons than because they would not bring real benefits to APL programs, control-structures are part of the daily routine of most of the APL programmers who attend APL conferences, so it's hardly a topic worth more than a five minute announcement. What is really sad is that Soliton is doing a lot of interesting work based on their Java interface and that they allowed themselves to commit public suicide by making believe the unaware audience that there's nothing more exciting in Sharp APL than ":if" and friends.

In fact, just like Soliton's show, the conference turned out to be a bit of a let-down, not because of a real lack of interesting topics that the community could have brought up, but because of the poor choice. I would not blame the organisers for this, since even a genius artist would have trouble shaping a masterpiece out of bad raw materials, and also because the program committee (I was part of it) did not do anything to improve the situation when they noticed that the content would not be up to standard.

Since we cannot change the past, it would be a good idea to at least try and make the best out of this unsatisfying experience and learn from our mistakes. In fact those who were disappointed by this year's conference should be looking forward to better occasions in the future. Maybe because of its youth, APL2000's user conference is still in the improving phase, despite its already good quality, and has pretty much substituted the SIGAPL conference in the heart of very many American users. Even Dyalog APL's users have started attending it, and would probably welcome a European event, similar in content and organisation, better if centred around their favourite interpreter. Does this mean that it is time to retire The (capital "T") APL conference? I don't think so, at least not without giving it another chance. I believe that fragmenting the community more than it already is would be detrimental to the future development of array oriented languages. It's only through the exchange of experiences and ideas that there can be progress. This is one of the reasons why I am very much in favour of expanding the scope of the conference to other array oriented languages: J, K and the old but new A+, which are closest to APL but also the other ones. Berlin2000 featured a few, academic but promising ones, and the paper by Paul Cockshott on Vector Pascal was the best research paper presented in Madrid2002. Now that the rest of the computing world, outside the laboratories of abstract research, is starting to see the benefits of the think-array philosophy when providing solutions for everyday business issues, now is the time to mix our experience, hopefully still supported by an ageless enthusiasm, with their new and fresh approach.

Quick Reference Diary

Date	Venue	Event
27-28 March, 2003	Helsinki, Finland	Forests in Finland 2 day meeting, details to be announced.
11-14 June, 2003	San Diego, USA	FCRC:APL2003 joint event See **Call for Papers** on page 6
November 2003	Naples, Florida	APL2000 User's Meeting. Date to be announced Location Naples Beach Hotel as usual.

Dates for Future Issues of VECTOR

	Vol.19 No.3	Vol.19 No.4	Vol.20 No.1
Copy date	6th Dec	7th March	20th June
Ad booking	13th Dec	14th March	27th June
Ad Copy	20th Dec	21st March	4th July
Distribution	January 2003	April 2003	July/Aug 2003

APL2003: Stretching the Mind!

FCRC: APL2003 Call For Papers

SIGAPL is pleased to announce that **APL2003** will be held in San Diego, California from Wednesday June 11 through Saturday June 14, 2003. This year we will participate in the 2003 *Federated Computing Research Conference.*

FCRC is an umbrella for over a dozen ACM-sponsored conferences. Other FCRC events span June 7-14, 2003. Quoting from the FCRC Web page:

> *The Federated Computer Research Conference (FCRC) assembles a spectrum of affiliated research conferences and workshops into a week long coordinated meeting held at a common time in a common place. This model retains the advantages of the smaller conferences, while at the same time, facilitates communication among researchers in different fields in computer science and engineering. Mornings of FCRC week will begin with joint plenary talks on topics of broad appeal to the computing research community.*

FCRC offers a broad canvas for the exchange of ideas, development experiences, and tools. Compare yours to the world! If you have an application, this is the place to compare notes with others who have done it in other ways, and bring their own perspectives and experience.

APL2003 provides us with the opportunity to showcase APL and APL applications to our professional peers; we believe a number of the other conferences at FCRC offer exceptional opportunity to exchange knowledge and ideas with our professional peers:

- EC'03: The Fourth ACM Conference on Electronic Commerce
- MOD/PODS: ACM SIGMOD/PODS 2003 Conference
- PLDI: ACM SIGPLAN Conference on Programming Language Design and Implementation
- PPoPP: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming
- PADS: Parallel and Distributed Simulation Workshop
- SPAA: ACM Annual Symposium on Parallelism in Algorithms and Architectures

If you want to see how APL really measures up, contribute your best experience, and see how others really react. Serve up the best you can at the food court of ideas from the APL kitchen.

The FCRC web site is located at http://www.acm.org/fcrc.

The SIGAPL conference committee invites you to submit papers for review on

- applications
- techniques
- education
- history
- language design and enhancement

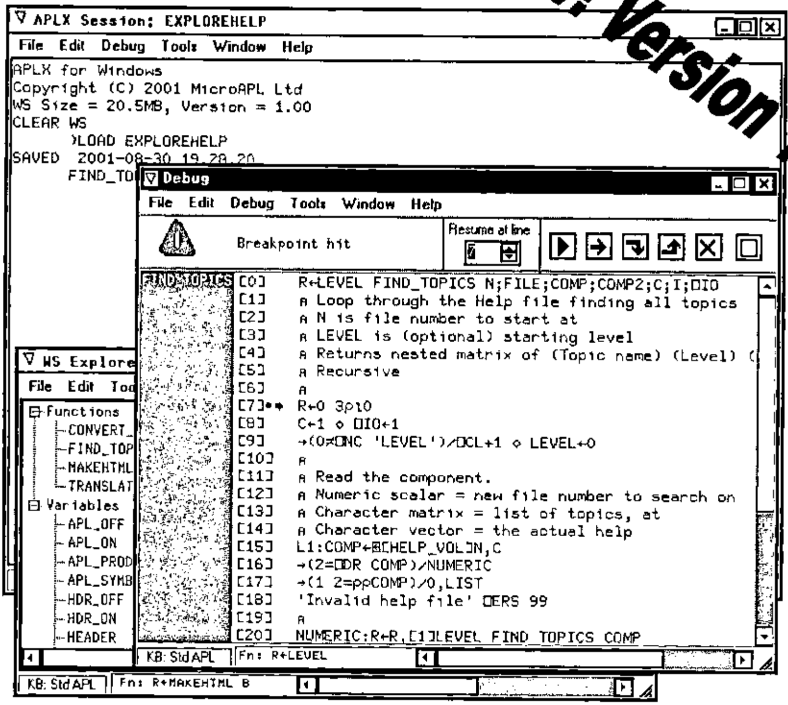
and other topics related to APL, A+, J, K, Matlab, Mathematica, Nial, S-Plus and other array programming languages. The users and advocates of different language that are heavily oriented toward array programming have many common values. APL2003 can be an event where we come together and face the future.

Themes you may wish to explore include (but are not limited to) are the following:

- Practical applications using arrays
- Interfaces between APL and other languages
- Array-based programming in the classroom
- GUI Programming using array-based languages
- APL on various platforms (Windows, Unix, Pocket PC)
- Similarities and difference among array programming languages

Please submit a 1-page abstract by January 31, 2003. Final drafts for review are due by March 1, 2003. We also invite those interested in preparing workshops, tutorials, poster sessions, panel discussions and other items of interest to contact Kevin Weaver, program chair:

Kevin Weaver · 14849 Horseshoe Trace · Wellington, FL 33414
(561) 790-6419 · fax: (561) 790-6425 · e-mail: kevin@modelfitness.com



```
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
```

```
[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 3ρι0
[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=ρρCOMP)/0,LIST
[18]    'Invalid help file' ⎕ERS 99
[19]    ⍝
[20]    NUMERIC:R←R,[1]LEVEL FIND_TOPICS COMP
```

BAA: Sustaining Members' News

MicroAPL Ltd

The new version 1.1 of *APLX*, which was previewed at APL2002 in Madrid, is now shipping on all platforms (Windows and MacOS, plus server editions for Linux, AIX, and Windows servers). The full list of new features is too long to detail here, but highlights include:

- Support for the 'structured control' programming keywords (`:If` etc).
- Support for using the double-quote as an alternative to the traditional single-quote character for delimiting APL character vectors.
- `⎕DISPLAY` and `)DISPLAY`, as well as a new Display window which shows you the structure of an APL array with just a click of the mouse.
- Powerful graphics drawing, bitmap and picture display, and animation facilities, using the `Draw` method.
- Support for Windows OCX/ActiveX controls, embedded OLE documents, and OLE servers. For example, you can control and exchange data with Excel, or include a Word document inside a `⎕WI` window and program it from APL. There is also a Control browser which allows you to browse through the external classes installed on your machine.
- The `maxsize` and `minsize` properties allow you to put constraints on `⎕WI` window sizes.
- The RichEdit control is now available under MacOS as well as Windows.
- A new keyboard-customisation dialog allows you to edit your own keyboard layout by dragging symbols on to the keyboard picture.
- Speed-ups to various primitives, especially matrix divide under Windows.
- Improved documentation, with fully searchable HTML help now available under MacOS as well as Windows. Context-sensitive help (using F1 or the Mac Help key) is also improved, recognising `⎕WI` keywords and not requiring the word at the cursor to be selected.

In addition, we have included the promised 'packager' feature which bundles your workspace with a free runtime edition of the interpreter. To make an executable application from your APL workspace, just choose Save As... from the File menu, and select 'Executable (packaged) ws' as the file type. (Under Windows, you also need a DLL, downloadable free of charge from our website).

APLX Version 1.1 runs on all Windows operating systems from Windows 95 through to XP, and on all MacOS versions from 8.6 onwards, including the new MacOS 10.2 'Jaguar'.

To find out more, you can download documentation on the new features, or the complete new manuals in PDF form, from http://www.microapl.co.uk/apl. Time-limited evaluation versions of the full product are also available.

Many of the version 1.1 improvements have been in response to the excellent feedback we have received from APLers around the world. Please keep the suggestions coming - the next major release will be version 2.0, which will include some unique new development-environment features as well as important language enhancements and new ⎕*WI* classes.

Causeway Graphical Systems Ltd

At Madrid, APL2C and Causeway agreed that it would be good for both of us to work more closely together. The APL2C engine is ideal for delivering Causeway's *GraPL.net* server product as a single lightweight DLL on the Windows platform, and also allows us to ship a command-line EXE file on any of the common Unix systems. The *RainPro* source can be automatically converted to run in APL2C, and then compiled and linked to make a COM server, which is much lighter to load than a full APL system with interpreter and workspace.

We also see the potential to offer APL developers a service which will take existing VS APL or APL2 code and wrap this as a single-file DLL with the appropriate COM wrapper automatically constructed. Before we can do this, we would like to prove the technology as thoroughly as we can, and the best way of doing this is to ship a good subset of *RainPro* and *NewLeaf* in this format. We will also be quietly working through the test examples in the ISO APL Standard (and Extended APL Draft Standard) which will ensure that the service we offer can reliably compile any standards-conforming (Gui-free) APL code.

Please do not expect any quick results from this project. We already have a single-file DLL version of *RainPro* which runs all our test charts correctly (and fast) but there is a long way to go before APL2C will be available as a fully-supported Causeway product for general-purpose use in compiling arbitrary APL2 code. We need to design the structures which will automatically generate the COM interfaces from your public functions, and test and retest the engine before we feel we can offer it for use to the APL community.

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

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 User Interface. 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. K.CoSy notepad nature, interactivity, and open K code vocabulary make learning Arthur Whitney's K far less daunting and far more productive than its raw console, or any external scripting method. If you are a client of Kx Systems , or are investigating the possibilities, Contact us. See CoSy/K/CoSy for more information.
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, algorithms, tree processing and design programs for craft work.
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 and Training	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) modelling.
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.
Lambent Technolgy	Consultancy	poa	APL programming, consulting & training; web design and construction.
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, 25 years experience. Custom software development, 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.

Right Seat Software	Vox Proxy	$199 (comm) $69.95 (edu)	Vox Proxy is authorware for PowerPoint(r) 2000 or 2002 which allows the use of Microsoft Agent Technology (3D talking animated characters) within slide shows. VP appears on PowerPoint's main menu and provides editing side-by-side with slides. Automated script-writing provides control of PowerPoint, allowing the use of characters for live presentations or fully-automated tutorials, demos, or training programs. Optional CD Prep program allows the user to create auto-starting CD's that will play on any version of PowerPoint or without PowerPoint.

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		Occasional meetings, APL Skills Database, Toronto Toolkit	

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@ca.inter.net
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 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
Lambent Technology Ltd	Stephen Taylor	81 South Hill Park, London NW3 2SS, UK. Tel: +44(0)20 7813 3786. Email: sjt@lambenttechnology.com
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	Optima House, Mill Court, Spindle Way, Crawley, West Sussex, RH10 1TT, 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 511, Botsford, CT 06404, USA. Tel: +1 (203) 270-9729 Email: aplbooks@earthlink.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
Right Seat Software, Inc.	Tom Atkins	1110 12th Street, Golden, CO 80401, USA. Tel:+1 303 278 2244. Fax: +1 303 278 6967. Email: info@voxproxy.com
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	P.O. Box 330, Excelsior, 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 Borealis Inc.	www.aplborealis.com
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
Lambent Technology	www.lambenttechnology.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.com/
Resources & Results	www.decision-support.net
RE Time Tracker Oy	www.rett.fi/
Right Seat Software, Inc.	www.voxproxy.com
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

Article I. Zark Newsletter Extracts

introduced by Jon Sandles

I hope you are all still enjoying these Zark reprints! Once again, we include a new crossword for your enjoyment and another classic APL problem. It would be interesting to see some new solutions to these problems, perhaps using Dyadic's dynamic functions, or some J or even K. Solutions will be printed in the next issue.

1. Utility Corner: Internal Rate of Return

(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.)

The term "Internal Rate of Return" (or IROR) is used to refer to the implied interest rate in effect from a given set of cash flows. Consider, for example, the following set of annual cash flows:

```
¯100 6 6 6 6 106
```

At the start of the first year, 100 is paid out. At the end of the next four years, 6 is paid in. 106 is paid in at the end of the fifth year. The IROR of this set of flows is 6% per year. To see this intuitively, view the cash outflow (`¯100`) as a deposit into a bank paying 6% per year. View the next four inflows of 6 as interest withdrawals at the end of subsequent years. View the final inflow of 106 as an interest withdrawal of 6 plus the return of the original 100 deposit.

To quantitatively verify that 6% is the IROR, the cash flows should sum to 0 when discounted to their present value or when accumulated to their future value using this rate:

```
      +/¯100 6 6 6 6 106×1.06*-⍳6
¯2.842170943E¯14
      +/¯100 6 6 6 6 106×1.06*⌽⍳6
¯4.263256415E¯14
```

Close enough to zero. Your task is to write a monadic function IROR that takes a vector of cash flows as its argument, one number per time period, and returns the internal rate of return, like so,

```
      IROR ¯100 6 6 6 6 106
 .06
```

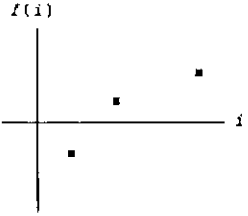
Our aim was to find the fastest possible IROR algorithm. The results surprised us. The most common method wasn't the fastest.

From a mathematical perspective, the goal is to solve for the variable *i* such that the value of the following formula is zero for a given vector of cash flows *CF*. (The symbols "*f(i)*" are read, "a function in terms of *i*.")

```
 f(i):     +/CF×(1+i)*-⍳⍴CF
```

Since this problem defies a direct algebraic solution, the only alternative is trial and error (called, "the method of successive approximations"). Try a value of *i*, say .08, in the expression and see how close the result is to zero. Then try another value, say 0.9, and see whether its result is closer to zero of further away. As you try more and more values, you'll get better at guessing values of *i* that return a value of *f(i)* that is close to zero.

In fact, you may find it helpful to plot the results. Plot the trial values of *i* along the X-axis, and the result from the expression along the Y-axis:



Graphically, *f(i)* describes a curve. The trial values of *i* and *f(i)* are points along that curve. From the graphic perspective, your job is to find where the curve intersects the i-axis, that is, to find the value of *i* where *f(i)=0*.

Four different approaches were suggested:

1. Extrapolate/interpolate, using a straight line *f(i)=a+(b×i)* fitted to the two best points so far. The "best" points are determined to be those closest to the i-axis, i.e. for which their values of *f(i)* are nearest zero. Draw a straight line through those points and determine the value of *i* where the line crosses the i-axis. That value of *i* becomes the next guess. Naturally, you need to make two "random" guesses to start the process.

2. Extrapolate/interpolate, using a quadratic curve `f(i)=a+(b×i)+(c×i*2)` fitted to the three best points. Just as the fitting of a straight line requires tow points, the fitting of a second degree curve (parabola) requires three points. The primitive APL function ⌹ simplifies the task of fitting the curve. Since the parabola crosses the i-axis at two points, only the solution nearest the other guesses is considered.

3. Extrapolate/interpolate, using a straight line that passes through the best point and has the same slope as the `f(i)` curve at that point. The slope of the `f(i)` curve at any point is the value of the derivative of `f(i)` with respect to `i` at that point. Computing the derivative of `f(i)` is a straightforward calculus operation:

   ```
   f'(i):      +/CF×(-⍳⍴CF)+(1+i)≠-1+⍳⍴CF
   ```

   Given `i` as the best guess so far, and `f(i)` as the function's value at that value of `i`, a small amount of algebra produces a formula for the value of `i` where the straight line crosses the i-axis:

   ```
   i':      i-(f(i)÷f'(i))
   ```

   This value of `i` is the next guess. This method of successive approximations is known as the Newton-Raphson method, and is the most common technique employed for solving the IROR problem. It requires only one guess to get started.

4. Extrapolate/interpolate, using a quadratic curve that passes through the best point and has the same slope as the `f(i)` curve at that point, and has the same rate of change in slope (second derivative) as the `f(i)` curve at that point. This approach extends the Newton-Raphson method to consider the second derivative as well as the first:

   ```
   f''(i):      +/CF×(-⍳⍴CF)×(-1+⍳⍴CF)×(1+i)≠-2+⍳⍴CF
   ```

   The subsequent algebra is considerably more complex, but the resulting curve does a better job of extrapolating or interpolating. Again, only one guess is needed to get started.

Each of these four approaches successively determines values of `i` that are better than the ones before. You stop the iterative process once you find a value of `i` for which the value of the function `f(i)` is "adequately" close to zero, say plus or minus `1E¯11`.

Which of these approaches is the fastest?

The advantage of approaches 3 and 4 is that they require a single guess to get started. The disadvantage is that the derivative (and second derivative) must be computed for each trial of `i`. The computation of the derivative (and second derivative) is more involved than the computation of the function. Fewer trials are needed, but each trial is slower.

The advantage of approaches 2 and 4 is that they use a parabola rather than a straight line when interpolating. Since a straight line is a special case of a parabola, the parabola can do no worse than the straight line and, in practice, requires about half as many trials. The disadvantage is that the calculations required at each iteration are more complex, and hence slower.

To find out which approach was fastest, we timed them on a 360-element vector of cash flows. In the process of timing, we noticed that the majority of the processing time was consumed by exponentiation — even with a math (floating point) coprocessor installed. To eliminate the exponentiation, we took advantage of the fact that the exponentiation was being done on successive integral powers, and replaced the exponentiation by cumulative products. To see how this is possible, note that the following two expressions produce equivalent results:

```
(÷1+i)*⍳360    ⍝ (origin 1)

×\360⍴÷1+i
```

Eliminating the exponentiation from all four approaches improved their speeds considerably. Here are the results of the timings (scaling the results so the fastest algorithm ran in 100 "units" of time):

	Using *	Using ×\	
IROR1	273	100	line from best 2 points
IROR2	252	102	parabola from best 3 points
IROR3	285	181	line from derivative (N-R)
IROR4	260	202	Parabola from 2nd derivative

You can see the simplest algorithm was the fastest! Also, the most common algorithm, IROR3 using * (Newton-Raphson), was the slowest. So much for mathematical sophistication.

As always, timings are affected by your hardware and your implementation of APL. You may want to time these functions on your own machine.

The functions used in the timings are listed below. In each function, the logic for both the exponential and cumulative product algorithms is included, though the exponential logic is commented out.

```
      ∇ R←IROR1 V;G;J;N;P;T;⎕IO
[1]    ⍝ Given a vector of cash flows V, one
[2]    ⍝ number per time period, returns the
[3]    ⍝ internal rate of return. Uses
[4]    ⍝ straight line interpolation.
[5]     ⎕IO←0
[6]     G←1.05 1.1 ⍝ Init guesses (5%,10%)
[7]    ⍝ T←-⍳⍴V ⍝ Periods for present values
[8]    ⍝ P←(G∘.*T)+.×V ⍝ Present values
[9]     N←⍴V ⍝ No. periods for present vals
[10]    P←V+.××⍀(N,2)⍴÷G ⍝ Present values
[11]   L1:R←(-/P×⌽G)÷-/P ⍝ Interp for rate
[12]    →(∨/1E¯11>|R-G)⍴L2 ⍝ Exit if ~ same
[13]   ⍝ P←P,V+.×R*T ⍝ Present val for rate
[14]    P←P,V+.××\N⍴÷R ⍝ Pres. val for rate
[15]    P←P[J←2↑⍋|P] ⍝ Pick 2 nearest zero
[16]    G←(G,R)[J] ⍝ ... & associated rates
[17]    →L1 ⍝ Try again
[18]   L2:R←R-1 ⍝ Convert to interest rate
      ∇
```

```
      ∇ R←IROR2 V;A;B;C;G;J;N;P;T;⎕IO
[1]    ⍝ Given a vector of cash flows V, one
[2]    ⍝ number per time period, returns the
[3]    ⍝ internal rate of return. Uses
[4]    ⍝ quadratic interpolation.
[5]     ⎕IO←0
[6]     G←1.075 1.05 1.1 ⍝ Guess 7.5,5,10%
[7]    ⍝ T←-⍳⍴V ⍝ Periods for present values
[8]    ⍝ P←(G∘.*T)+.×V ⍝ Present values
[9]     N←⍴V ⍝ No. periods for present vals
[10]    P←V+.××⍀(N,3)⍴÷G ⍝ Present values
[11]   ⍝ Coefficients of quadratic:
[12]   L1:C←PŽ(G×G),G,[0.5]1
[13]   ⍝ NOTE Ž sometimes fails on DOM ERROR
[14]    A←C[0]
[15]    B←C[1]
[16]    C←C[2]
[17]   ⍝ Get roots from quadratic formula:
[18]    R←((-B)+1 ¯1×(0⌈(B×B)-4×A×C)*0.5)÷A+A
[19]    R←R[>/|R-G[0]] ⍝ Pick the best root
[20]    →(∨/1E¯11>|R-G)⍴L2 ⍝ Exit if ~ same
[21]   ⍝ P←P,V+.×R*T ⍝ Present val for rate
[22]    P←P,V+.××\N⍴÷R ⍝ Pres. val for rate
[23]    P←P[J←3↑⍋|P] ⍝ Pick 3 nearest zero
[24]    G←(G,R)[J] ⍝ ... & associated rates
[25]    →L1 ⍝ Try again
[26]   L2:R←R-1 ⍝ Convert to interest rate
      ∇
      ∇ R←IROR3 V;D;F;G;N;T;T1;U;V1;⎕IO
[1]    ⍝ Given a vector of cash flows V, one
[2]    ⍝ number per time period, returns the
[3]    ⍝ internal rate of return. Newton-
[4]    ⍝ Raphson: for a function f(x),
[5]    ⍝ whose derivative is f'(x), the next
[6]    ⍝ guess should be x-f(x)÷f'(x). That
[7]    ⍝ is, use the slope at x to extrap.
[8]    ⍝ to where f(x) is zero.
[9]    ⍝     f(x) = +/(G*T)×V
[10]   ⍝    f'(x) = +/T×(G*T-1)×V
[11]    ⎕IO←0
[12]   ⍝ T←-⍳⍴V ⍝ Periods for present values
[13]   ⍝ V1←T×V ⍝ Product for derivitave
[14]   ⍝ T←T-1 ⍝ Exponents for derivative
[15]    N←⍴V ⍝ No. periods for present vals
[16]    T1←¯1-⍳N ⍝ Product for derivative
[17]    G←1.1 ⍝ Initial guess (10%)
[18]  L1: ⍝ D←V1+.×U←G*T ⍝ f'(x)
[19]   ⍝ F←V+.×¯1↓1,U ⍝ f(x)
[20]    F←+/U←V××\N⍴÷G ⍝ f(x)
[21]    D←T1+.×U÷G ⍝ f'(x)
[22]    R←G-U←F÷D ⍝ New rate
[23]    →(1E¯11>|U)⍴L2 ⍝ Exit if ~ same
[24]    G←R ⍝ Make this the next guess
[25]    →L1 ⍝ Try again
[26]  L2:R←R-1 ⍝ Convert to interest rate
      ∇
```

```
      ∇ R←IROR4 V;A;B;C;D;F;G;N;S;T;T1;T2;U;V1;V2;⎕IO
[1]    ⍝ Given a vector of cash flows V, one
[2]    ⍝ number per time period, returns the
[3]    ⍝ internal rate of return. Newton-
[4]    ⍝ Raphson extended to the 2nd deriv.
[5]    ⍝ For a function f(x), whose deriv.
[6]    ⍝ is f'(x) and 2nd deriv. is f"(x),
[7]    ⍝ the next guess should be the root
[8]    ⍝ of the quadratic equation
[9]    ⍝     0 = f(Z) = (A×Z*2) + (B×Z) + C
[10]   ⍝ where:
[11]   ⍝    A = f"(x)÷2
[12]   ⍝    B = f'(x)-f"(x)×x
[13]   ⍝    C = f(x)+(-x×f'(x))+(x*2)×f"(x)
[14]   ⍝ and:
[15]   ⍝     f(x) = +/(G*T)×V
[16]   ⍝    f'(x) = +/T×(G*T-1)×V
[17]   ⍝    f"(x) = +/T×(T-1)×(G*T-2)×V
[18]    ⎕IO←0
[19]   ⍝ T←-⍳⍴V ⍝ Periods for present values
[20]   ⍝ V1←T×V ⍝ Product for derivitave
[21]   ⍝ T←T-1 ⍝ Exponents for derivative
[22]   ⍝ V2←T×V1 ⍝ Product for 2nd deriv
[23]   ⍝ T←T-1 ⍝ Exponents for 2nd deriv
[24]    N←⍴V ⍝ No. periods for present vals
[25]    T1←¯1-⍳N ⍝ Product for derivative
```

```
[26]    T2←T1-1 ⍝ Product for 2nd deriv
[27]    G←1.1 ⍝ Initial guess (10%)
[28]   L1: ⍝ S←V2+.×U←G*T ⍝ f"(x)
[29]   ⍝ D←V1+.×U←¯1↓(÷G),U ⍝ f'(x)
[30]   ⍝ F←V+.×¯1↓1,U ⍝ f(x)
[31]    F←+/U←V××\Nρ÷G ⍝ f(x)
[32]    D←+/U←T1×U÷G ⍝ f'(x)
[33]    S←T2+.×U÷G ⍝ f"(x)
[34]    A←S÷2
[35]    B←D-S×G
[36]    C←F-G×D-G×A
[37]    ⍝ Get roots from quadratic formula:
[38]    R←((-B)+1 ¯1×(0⌈(B×B)-4×A×C)*0.5)÷A+A
[39]    R←R[>/|R-G] ⍝ Pick best root
[40]    →(0.00000000001>|R-G)ρL2 ⍝ Exit if ~ same
[41]    G←R ⍝ Make this the next guess
[42]    →L1 ⍝ Try again
[43]   L2:R←R-1 ⍝ Convert to interest rate
      ∇
```

Some final notes. These IROR functions have very specific behaviour. You may want to modify them in a number of ways:

1. The iterative process is halted when the new guess (value of *i*) varies from the previous guess by less than 1E¯11. You may require more or less precision.

2. The process begins with one to three interest rate guesses in the range of 5 to 8 percent. If you have better knowledge of what the unknown interest rate is, you may adjust these initial rates to move them closer to the unknown rate. In this way, you may be able to shorten the process by an iteration or two. In our trials, we found that the process required the same number of iterations regardless of the initial rates ... unless the initial rates were very close.

3. In some IROR problems, the interval between cash flows is not uniform. The second cash flow may code in 45 days, the third in 425 days, the fourth in 450 days, and so on. If this is the case, the time period component of the formulas must be changed from the vector ιρ*CV* to a vector representing fractions of years from the first cash flow to each subsequent cash flow. This change, while straightforward, eliminates the possibility of using cumulative product (×\) in place of the slower exponentiation (*).

2. Limbering Up: Effective Date Searching

(The purpose of this column is to work some flab off you 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 two numeric vectors of the same length, *IDS* and *PRICES*. *IDS* is a vector of distinct product ID codes for all available products; *PRICES* is a vector of corresponding prices for these products. Given a subset list of product ID codes, say *LIST*, you can easily determine their corresponding prices:

```
  PRICES[IDSιLIST]
```

Suppose prices change over time. Each prices, for a given product, has an associated effective date. For a given product, there may be many different prices, each with a different effective date. To determine the price of a given product, you need to specify not only the product's ID code, but also the date as of when you want its price.

Here's an illustration

Product Id	Price	Effective Date
201	2.65	1993/01/01
	2.75	1993/07/01
	2.89	1993/09/31
202	15.50	1993/01/01
	16.75	1993/10/15

Your task is to design and define a utility function that takes a list of product Ids and corresponding dates, and returns the prices for those products on those dates. We're looking for two things: clean design, and program efficiency.

Send your solution 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.

Ed: When putting this item together I noticed that *IROR1*[10] would be more efficient if the calculation of present values used decode (⊥) rather than either the exponentiation or the cumulative product techniques. Much to my surprise and horror, the decode algorithm produced a different result to the cumulative product code, but the same result as the exponentiation code. The following demonstrates the problem:

```
      IROR1 {(⌊¯.05×+/⍵),⍵}?359⍴1000
IROR1[10]
      V+.××⍀(N,2)⍴÷G      ⍝ Cumulative product method
¯53.61702776 ¯4305.479388
      (2 1⍴÷G)⊥⌽V         ⍝ Decode method
¯56.29787915 ¯4736.027327
      (G∘.*-⍳⍴V)+.×V      ⍝ Exponentiation method
¯56.29787915 ¯4736.027327
```

I can only assume that the cumulative product accumulates errors as the numbers are multiplied together, whereas the other methods do not. In spite of this difference the final IROR result seems to be the same in each case.

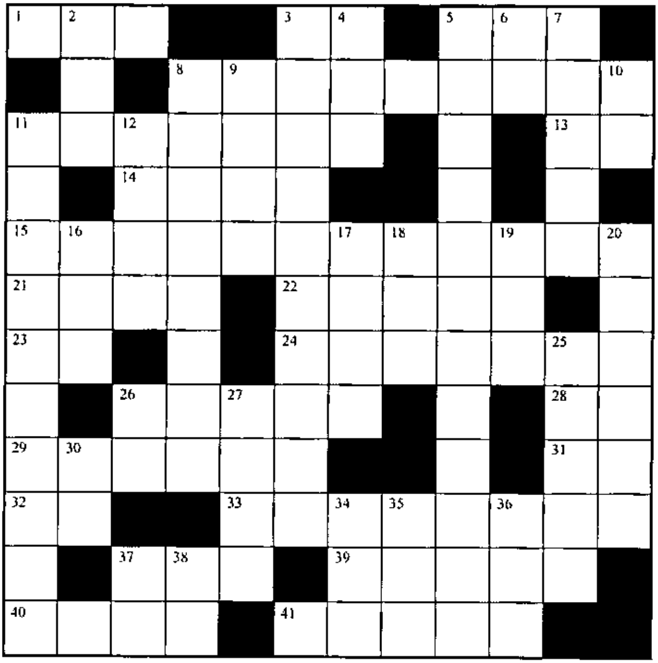
Reprinted with kind permission from Zark APL Tutor News, a quarterly publication of Zark Incorporated, 23 Ketchbrook Lane, Ellington, CT06029, USA.

Solution for 19.1 and New Crossword

Solution to Crossword in 19.1

■	M	⌊	S	M	■	■	→	L	×	⍳	■
(	N	,	⍴	A	)	⍴	A	■	A	⌈	N
0	=	B	-	T	S	D	■	-	-	/	V
∨	G	■	J	∊	I	,	K	■	B	|	E
.	5	⋆	×	S	■	L	~	0	■	N	C
≠	■	1	⌽	E	X	P	R	■	⌈	■	[
T	■	■	N	L	■	■	,	V	T	■	⍋
R	⍕	M	■	■	A	×	1	⊥	×	⌿	N
)	W	S	I	D	■	/	■	1	6	■	V
/	■	+	\	-	N	⍴	1	■	■	-	E
T	W	?	T	W	■	M	■	■	∧	\	C
R	A	T	E	S	[	V	⍳	⌈	/	V	]

New Crossword



Across

1. `(~A∊B)/A`
3. Where in V is ...
5. `,?N`
8. Do any of the items in nested NST match the array NR?
11. `∨/B[29+⍳(⍴B)-29]`
13. Magnitude of D
14. Array of shape T5, filled with the elements of Z
15. Any array ABS, with all numbers replaced by 0s and characters by blanks
21. Correspondingly true in N and PA
22. Which elements of V match the corresponding elements of the vector in the enclosed scalar CP?
23. C, except ...
24. Is the NVLth item of vector A the same as the array E?

26. Which item of N first matches CV?
28. (× / ρ *L*) ρ *L*
29. Rotate the elements of vector T left Q places, format them, and stick the label V on the front
31. ... replicate A
32. The keys for ∊ |
33. Flag the items of nested NST that match the array NR
37. The length of vector EV
39. The next higher integer from NNMS
40. The first element of X as a scalar
41. Never forget to _____

Down

2. ∧ / ,
3. *VECTS*[*VN*↓⍳*Z*]
4. 1, 2, 3, ..., RB
5. Flag the character matrix items of nested NV that contain the string CV
6. A random scalar from the set ⍳*N*
7. The remainder of *JB*÷*NS*
8. The elements of Q are first located in which elements of the 25th item of A?
9. Nine ones, as a nested scalar
10. The keys used for ~⌊
11. Does VEC exist in the workspace?
12. TOP=0 for Boolean TOP
16. *I* > for Booleans
17. Is V a scalar?
18. The shape of the array within the enclosed scalar N
19. The language J's older brother
20. An array with empty shape
25. Catenate together the items of nested vector NS
26. Format to N decimal places
27. NV, flipped and enclosed
30. The elements of M as a vector
34. The larger of T or S
35. The depth (level of nesting) of NA
36. Create a nested scalar from array ME
37. Rank of ...
38. ⎕__ is used to erase objects

J-ottings 34: Greed – Patterns for the Imminent Collapse of Western Capitalism

by Norman Thomson

In a communication following J-ottings 33, Ken Iverson pointed out that $x/(1-x-x^2)$ is a generating function for the Fibonacci numbers which in J terms is

```
  fibfn=.0 1&p. % 1 _1 _1&p.
```

Rewriting the generating function as

$$x(1-x(1+x))^{-1}$$

the binomial theorem gives the series expansion

$$x(1+x(1+x)+x^2(1+x)^2+x^3(1+x)^3+\dots$$

Write the coefficients of the various binomial coefficients as rows of a table :

x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
1								
1	1							
	1	2	1					
		1	3	3	1			
			1	4	6	4	1	
				1	5	10	10	5
						..	..	

Adding down the columns should make it clear how the Fibonacci coefficients arise. But why do all this work when J will do it for you using the t. adverb to give the best fitting polynomial of given degree, say the 13th. ?:

```
      fibfn t. i.14
  0 1 1 2 3 5 8 13 21 34 55 89 144 233
```

which is exactly equivalent to the expression in J-ottings 33 :

```
      f=.(,+/@(_2&{.))^:12(0 1)
```

Yet another interesting Fibonacci fact is that every positive integer can be written as the sum of a series of non-consecutive Fibonacci numbers, non-consecutive because every sum of a consecutive pair can immediately be replaced by the next higher Fibonacci number – this is the fundamental Fibonacci property. To obtain such a sum, start with a general verb which transforms any given number into the "highest value below" in a numeric list :

```
      hvb=.>./ @(>:#])   NB. highest value below
      100 hvb f
  89
```

Define

```
      gap=.-hvb&f
      gap 100
  11
```

Both the 89 and the 11 are required, the first to be stored, and the second to be processed further, so define a verb which partitions an integer in this fashion:

```
      Fgap=.(hvb,[-hvb)&f     NB. partition into fib+gap
      Fgap 100
  89 11
```

Using & makes f into a pseudo-constant which would be changed only if a shorter or longer Fibonacci series was required. Also, the double computation of the verb hvb in Fgap is inherently displeasing – this can be circumvented by rewriting the slightly less transparent verb :

```
      Fptn=.{:@(>:#(],.-))    NB. Fibonacci partition
      100 Fptn f
  89 11
```

following which Fsum remembers the left hand part of the list and processes the gap on the right :

```
      Fsum=.}:,Fptn&f@{: NB. single step process
      Fsum 89 11
  89 8 3
```

8 and 3 are both Fibonacci numbers, and so 100 = 89 + 8 + 3.

This process can be repeated as long as necessary using a recursive verb

```
      Fsumr=.($:@Fsum)`]@.(e.&f@{:)
      Fsumr 100
  89 8 3
```

which says in effect "go on reducing the gap until you reach a Fibonacci number".

100 (= 89 + 8 + 3) can also be expressed in a binary form

```
      Fnum=.|.@(2&}.@(f&e.)@Fsumr)
      Fnum 100
  0 0 1 0 0 0 0 1 0 1 0 0
```

call such numbers Finary numbers, say – they will be revisited later.

A characteristic of the above algorithm is that at every step the Fibonacci number giving the smallest gap was grabbed, a characteristic of greedy behaviour as exemplified alike by small boys sharing cakes and fat-cat directors raiding their company takings.

The term greedy algorithm is used generically to describe a range of algorithms which share this general characteristic. It arises sufficiently often to merit description as a pattern, a word which has become a technical term in the vocabulary of Object Oriented programming to mean any common way of doing things – more general than an idiom or a phrase, but not large enough to be considered as a piece of software architecture.

"Patterns" seems a good way of describing general techniques which emerge intuitively in J programming through recurrent use and reinvention. For example, the greedy technique apparent in Fsumr above might also have emerged in programming a distance reducing route starting from a given town and visiting all other towns represented in a given distance table. To be specific suppose that a distance table for four towns is

```
    0  1  5  3
    1  0  7  6
    5  7  0 10
    3  6 10  0
```

The routing problem is not altered by subtracting 11 from all the non-zero entries in the table and making the objective maximization rather than minimization. So define m as

```
    0 10 6 8
   10  0 4 5
    6  4 0 1
    8  5 1 0
```

Starting at town 0, initial greed says find the route which gives the highest reward from town 0. This is the route to town 1, from which the next greedy step is to go from town 1 to town 3 since 5>4, and the route is completed by visiting 2, thereby adding another 1 to the reward, a total of 16.

To preserve indexing it is prudent not to reduce the distance table at each recursive step but rather to reduce the reward to 0 for steps which proceed to towns already visited. Thus if the "route so far" is 0 2 1, the next town is identified by its index as the one offering the highest reward in row 1 after avoiding revisits, which requires that items 0 and 2 in row 1 must first be amended to 0 by `0(0 2)}1{m`. The index of the next town is generated greedily by

```
   imax=.i.>./          NB. index of maximum value
   ntg=.dyad define     NB. next town in greedy chain
imax 0(}:x.)}({:x.){y.
)
   0 2 1 ntg m
3
```

As with `f` above, a distance table would be unlikely to have frequent changes and so it makes sense to build it in as a pseudo-constant `m` :

```
   ontg=.ntg&m          NB. row index of optimum next town
   ontg 0 2 1
3
   ontgr=.]`($:@(],ontg))@.(4&~:@#)
   ontgr 0
0 1 3 2
```

A note should be made that a change in the distance matrix might require a change in the constant 4.

There is an important distinction between these two manifestations of greed. In the first, the representation of a positive integer as a sum of Fibonacci numbers can be proved to be unique, and so *any* algorithmic pattern would produce the same result. This is not true in the second example for which the route 0 3 2 1 has the value of 17, demonstrating that greed is not always the best policy!

It should also be stressed that it is recursion in the presence of maximum which makes these patterns greedy, not recursion alone. Simple recursion is a generalisation of the greedy pattern. This can be illustrated by venturing even further into Fibo-land and constructing a Finary adder. First assume that an ordinary binary adder is available:

```
   badd=.+&.#.                NB. binary addition
   1 0 1 badd 1 0 0
1 0 0 1
```

The main difference between binary and Finary numbers is that Finary numbers never contain consecutive 1s, and so if two consecutive 1s were ever to turn up in an intermediate calculation these would be immediately resolved into a single 1 at the level of the next higher digit. A verb to detect such a state of affairs is :

```
   find11=.0&,@(2&(*./\))     NB. find sublist 1 1
   b                          NB. representation of 148
1 0 1 0 1 1 0 1 0 1
   find11 b
0 0 0 0 0 1 0 0 0 0
```

Simple binary addition of these two lists makes the necessary adjustment :

```
    rep11=.badd find11    NB. replace with 1 0 0
    rep11 b               NB. representation of 148
1 0 1 1 0 0 0 1 0 1
```

Having made one such replacement it is possible that, as in the case above, the new 1 produces a further pair of consecutive 1s, which in turn may generate a ripple effect through the whole Finary number. Adopting the recursive pattern again, define :

```
   rep11r=.]`($:@rep11)@.(+./@find11)
   rep11r b               NB. representation of 148
1 0 0 0 0 0 0 1 0 1
```

Enough is now in place for a verb to add 1 to a given Finary number :

```
   Fadd1=.rep11r@(1&badd)
   Fadd1 1 0 1 0 1        NB. add 1 to Finary 12
1 0 0 0 0 0
```

To perform a general Finary addition, e.g. 1 0 1 0 Fplus 1 0 0 0 0 1, observe that the recursive pattern requires that the data is in the form of a single argument. One technique in the present case is to keep on applying `Fadd1` to one of the summands until the value in a counter matches the other, hence a suitable data format for the above sum is

```
    u=.0;1 0 1 0;1 0 0 0 0 1 NB. Finary 0(cntr),7,14
```

and a single step of the addition process is :

```
    Fplus1=.Fadd1&.>@(2&{.),{:     NB. Add 1 to each of 0,7
    Fplus1 u
 +-+---------+-----------+
 |1|1 0 0 0 0|1 0 0 0 0 1|    NB. Finary 1(cntr),8,14
 +-+---------+-----------+
```

Applying the general recursive pattern yet again leads to

```
    Fplus=.($:@Fplus1)`(>@(1&{))@.({.-:{:)
    Fplus u
 1 0 0 0 0 0 0
```

The problems above may in themselves be academic, but the use of patterns themselves is a highly practical programming matter. Using patterns is matter of design; in the examples described above, the generic pattern can be informally described as

fnr = fnr (fn), that is, a recursive verb defined as "itself applied to the result of a matching single step verb". Many, although certainly not all, programming problems at this level are susceptible to this design pattern. To use such a pattern, three questions must have clear answers:

1. Can I describe a single step verb whose result is the input to the next step?
2. How do I know when to stop the recursion?
3. What do I want to happen when it does?

What happens below the level of fnr = fnr (fn), for example whether code is tacit, explicit or a mixture, is a matter of implementation. For example, in the last case, an iterative design might have led to an implementation something like

```
    Fplus=.dyad define
 s=.0 [ r=.y.
 while.(-.s-:x.) do.
 r=.Fadd1 r [ s=.Fadd1 s end.
    1 0 1 0 Fplus 1 0 0 0 0 1           NB. 7 + 14
 1 0 0 0 0 0 0
```

Finally for those of you whose sensible habit is always to skip to the end, thereby cutting out all the turgid stuff in the middle, this article has been all about Fibonacci, patterns and greed, but the greatest of these is ... (reader to complete)

```
 ilast=.<:@(#-i.&1@(1 1&E.&|.))
                  NB. index of lowest order 1 1
rep=.monad define NB. replace lowest 1 1 with 1 0 0
i=.ilast y.
r=.0(i-0 1)}y.
if. i=1 do.r=.1,r
else. r=.1(i-2)}r end.
add=.(_2&}.),(_2&{.)@(1&+&.#.)
```

Reviews and Conference Reports

Dyalog APL for Windows CE (beta) A Hacker's First Impression

by Ray Cannon (email: ray_cannon@compuserve.com)
22nd September 2002

A new version of Dyalog APL was announced at the Dyadic Vendor forum at APL 2002 in Madrid. What makes it so special is that it is a FULL copy of Dyalog APL, it's expected to "cost as little as $50" and runs on the latest Pocket PCs.

A "Pocket PC" is a Personal Digital Assistant (PDA) running Microsoft Pocket PC 2002 software (Windows CE 3.0, Pocket Outlook, Pocket Word and Pocket Excel).

So two weeks ago I bought myself a HP Jornada 568, a Pocket PC with a 206MHz StrongArm processor, 64 MB ram, 32 MB flash rom, a 320×240 16-bit reflective TFT colour display, and type 1 Compact Flash port. I downloaded a beta copy of "Pocket APL" from Dyadic's web site, and have been playing with it in my free time ever since.

Keyboard

Most Pocket PCs use a stylus and virtual keyboard, but physical keyboards are available as add-ons for many makes, which allow one to "thumb type" (as against "touch type").

I do not yet have such a keyboard for my Jornada, so cannot tell you if it works in conjunction with the APL keyboard. However, since the virtual keyboard, when displayed, takes up about 1/3 of the screen, it would be a real asset if it does work, even if I still type with the stylus!

Using the stylus takes me back 30 years to my "ROLF HARRIS STYLOPHONE " days. On the Stylophone, you could not play chords, only single notes. So it is with the Pocket PC, "computer keyboard chords" (such as "rho" = <ctrl>r) are not possible, so are entered via a sequence of single strokes, <ctrl> then <r>.

(I wonder if I can turn my Pocket PC into a Stylophone via Dyalog APL?)

Installation

Installation of the beta copy of Dyalog (and later, a beta upgrade) was simple and painless. After installing APL I was in such a hurry to use it, that I failed to notice that the newly installed "Pocket APL Keyboard" had not been selected. So I wasted half an hour wondering how to type in APL. Once I had realised that I needed to select this second "keyboard", I was off. Moral – RTFM.

Having seen a demo of "Rain in Spain" I knew it would run many Dyalog APL workspaces, without problem. (Pocket APL will run the 1996 shareware version of Adrian Smith's *Rain,* supplied as an "outer product" with Dyalog 7.2, as demonstrated by Dyadic on the last day of the 2002 APL conference in Madrid.)

However, I also knew that the "Windows CE API", did NOT contain the same set of DLLs as the "Win32 API" as used by Win 95/98/Me/2000 etc., so, as an initial project, I decided to convert all my utility functions that used DLLs to also work under CE wherever possible.

Chicken or Egg?

The first problem I had to overcome was "How does code know if it's running under 'Win32 API' or 'Win CE API'?"

The API function "GetVersionEx" that is available under both Win32 and WinCE returns suitable information so one can make this choice. However, the name of the DLL it resides in is different. So the ⎕NA code needed to call it, needs to know which OS it is running, so one needs to call the API GetVersionEx.....

OK, a different approach is required, let's see what APLVersion gives us.

```
      '.' ⎕wg 'APLVersion'
Windows for Pocket PC  10.0.0  P  Development
```

Note, the 3rd element of APLVersion is "P" compared to "W" for the Win32 API, and "M" (Motif) from Unix based machines. (In the first beta release, APLVersion returned "CE" rather than "P" but this was changed to prevent length errors in running existing code, that some beta testers have reported.)

```
     ∇PoW←{⍝ Pocketpc or Win32 API?
      ⍝Returns  ⍵ if APLVersion is W else ⍺
         ⎕ML←0
         'W'=⊃⊃3⊃'.'⎕WG'APLVersion':⍵
         ⍺   ⍝ Pocket PC or Motife etc }
     ∇
```

Now please note that I have /am playing with BETA software, and have already received one update which fixed/resolved several issues with the original release. In addition to APL related issues, the update has greatly improved the input facilities available to the "developer" e.g. "Auto complete". Dyadic are acting on the feedback from the beta testers. So be warned, the final production release may well have significant changes from the current software I am testing.

Now, on a full-sized PC, I find "Auto complete" a real distraction, and I turn it off whenever possible. However, when armed with only a "pointed stick" and a "virtual keyboard" the "Auto complete" feature is a real boon. You will do almost anything to save a few "key-strokes" or should I say "stick-taps".

As implemented by Dyadic, the "Auto complete" works very well. I have adjusted its parameters so it does not guess until you have typed 2 characters. I found the default of 1 character just not worth the effort. (It's a bit like that old TV quiz "I'll name that tune in one", sheer luck if it is correct.)

The Core of the Kernel

Depending on platform, Windows APIs can handle text as either single-byte (ASCII) or multi-byte (normally 2-byte – UNICODE). To allow for this, many API functions which handle text can have two forms, and their names have either a trailing "A" (ASCII) or "W" (wide). Windows CE 3 supports UNICODE only, hence the API functions end with "W" not "A".

On a Pocket PC most of the available APIs are to be found in COREDLL. Whereas under Win32, the APIs there are spread out a lot more; in Kernel32, User32, Gdi32, Advpi32 to name just a few.

So here is a function to return the Windows version information:

```
      ∇ r←GetVersion;get;val;dll;dos;build;ex;rc;str;txt;⎕IO
 [1]    ⍝Return OS major, minor, build, platform/dos
 [2]    ⎕IO
 [3]    rc←0
 [4]    'ex'⎕NA{
 [5]       ⍵=1:'I kernel32|GetVersionExA ={I4 I4 I4 I4 I4 T[128]}'
 [6]       ⍵=2:'I coredll|GetVersionExW ={I4 I4 I4 I4 I4 T[128]}'
 [7]    }2 PoW 1
 [8]    :If 3=⎕NC'ex'
 [9]        str←148 0 0 0 0(128⍴' ')
 [10]       rc str←ex,⊂str
 [11]   :End
 [12]   :If rc
 [13]       str[4]←16⊥(4⍴16)⊤str[4]
 [14]       r←str[2 3 4 5]
 [15]       txt←6⊃str
 [16]       txt←(+/∧\txt≠⎕AV[1])↑txt
 [17]       r,←⊂txt
```

```
    :Else
        r←0 0 0 0 ''
    :End
    ⍝
    ⍝INFO for 32 bit application (ie not 16bit win 3.x)
    ⍝Win NT3.51       - 3 51  ?     ?
    ⍝Win NT4 (sp6)    - 4  0  1381  2 Service Pack 6
    ⍝Win 95  (OSR2)   - 4  0  1111  1 B
    ⍝                 - 4  0  1111  1 C
    ⍝                 - 4  0  1212  1 B
    ⍝                 - 4  0  1212  1 C
    ⍝Win 98           - 4 10  1998  1
    ⍝Win 98SE         - 4 10  2222  1 A
    ⍝Win ME           - 4 90  3000  1
    ⍝Win 2000 (sp3)   - 5 0   2195  2 Service Pack 3
    ⍝Win XP   (sp1)   - 5 1   2600  2 Service Pack 1
    ⍝Win CE 3.0       - 3 0  11171  3 ''
  ∇
```

So far, I have converted about 40 ⎕NA calling functions, and have had no difficulty with the APL. That's not to say that the Pocket PC is a great APL development environment, it's not. But the problems are with the limited resources available on the Pocket PC, not the APL.

The way I have been working is to modify a function on my desktop PC, and after testing that the code works, saving the workspace. This workspace is then transferred across to the Pocket PC. After loading this workspace on the Pocket PC, I test it again. If necessary, I can fix minor bugs and typos in the APL code, on the Pocket PC, and then transfer the workspace back to my desktop.

Finally, here is the code to return the version number of a DLL/EXE. Be aware that Dyadic only recently introduced support for "VerQueryValue" so running DllVersion on Dyalog.exe will only return a result for version 9.0.3 with a patched date some time after 22nd March 2002. (I do not know exactly when Dyadic introduced it.)

```
∇ version←DllVersion file;Aloc;Free;Lock;Ulok;Size;Info
   ;Valu;copy;size;hndl;addr;buff;ok
  ⍝ Get version number of DLL
  :If 0 PoW 1
      'Aloc'⎕NA'u kernel32GlobalAlloc u  u'
      'Free'⎕NA'u kernel32GlobalFree u'
      'Lock'⎕NA'u kernel32GlobalLock u'
      'Ulok'⎕NA'u kernel32GlobalUnlock u'
      'Size'⎕NA'u versionGetFileVersionInfoSizeA <0T >u'
      'Info'⎕NA'u versionGetFileVersionInfoA <0T u u u'
      'Valu'⎕NA'u versionVerQueryValueA u <0T >u >u'
      'copy'⎕NA'dyalog32.C32MEMCPY >u[] u u'
  :Else
      'Aloc'⎕NA'u coredllLocalAlloc u  u'
      'Free'⎕NA'u coredllLocalFree u'
      Lock←{⍵}
      Ulok←{1}
      'Size'⎕NA'u coredllGetFileVersionInfoSizeW <0T >u'
      'Info'⎕NA'u coredllGetFileVersionInfoW <0T u u u'
      'Valu'⎕NA'u coredllVerQueryValueW u <0T >u >u'
      'copy'⎕NA'coredllmemcpy >u[] u u'
  :End
  :If ×size←⊃Size file 0                   ⍝ Size of info.
  :AndIf ×hndl←Aloc 0 size                 ⍝ Alloc memory.
      :If ×addr←Lock hndl                  ⍝ Lock memory.
          :If ×Info file 0 size addr       ⍝ Version info.
              (ok buff size)←Valu addr'\' 0 0
                                           ⍝ Version value
              :If ok
                  buff←copy(size÷4)buff size    ⍝ Copy info
                  version←⊃,/(⊂2/2*16)⊤¨2↑2↓buff
                                          ⍝ Encode version
              :EndIf
          :EndIf
          ok←Ulok hndl                     ⍝ Unlock memory.
      :EndIf
      ok←Free hndl                         ⍝ Free memory.
  :EndIf
∇
```

Points to note relating to Pocket APL:

- Dyalog32.DLL was not shipped with the beta version, but since "coredll" contains a both "memcpy" and "strncpy" (note lower case spelling), this is not actually a problem.

- Pocket PC does not support Global memory calls, so these are replaced by Local memory calls.
- Local memory does not and cannot be "locked" so these calls are replaced by simple function calls returning appropriate results.

Conclusion

Dyalog Pocket APL is NOT a toy, nor a "cut down" version of their desktop product, but a real, state of the art, extremely cheap, full-blown APL, that runs in a PC that literally fits in your pocket. It can handle workspaces 3 times the size of the largest workspace I can use on the mainframe APL systems I still support, and the 256MB of Compact Flash memory I have compares well with the 300 MB of mainframe disk storage I have allocated.

With Pocket APL, the APL character set is NOT an issue, Pocket PCs are Unicode not ASCII based, and with the virtual keyboard, I can see APL characters as I type for the first time in 7 years!

I believe Pocket APL can give the APL community a new lease of life. It gives APL developers a flying start into the Pocket PC market. We can re-sell our existing products in a brand new environment. We can develop new products for a new target audience using our existing tools, with almost no learning curve.

To paraphrase the chancellor of the exchequer at the end of his budget speech:

> "I can commend this APL to the community."

If You Go Down to the Woods Today ... First Experiences with PocketAPL

by Adrian Smith (adrian@grapl.com)

First Encounter (Madrid 2002)

This must have been the best-kept secret at APL2002. There had been rumours of a "Project-X" in the Basingstoke labs for several months, but none of us guessed the shape and size of the egg that the Ducks had been lovingly incubating. When the chick was finally allowed to hatch, it generated just the "Wow!" reaction that Dyadic must have been hoping for.

Just as Eric Iverson had shown J to a fascinated BAA Meeting a couple of years ago, John Daintree was able to demo a fully-fledged pocket application that exploited the array-power of APL, and was moreover a useful thing to have in your jacket pocket. His "Mornington Crescent" solver used a publicly available topology of the London Underground to allow users to tap-select on their current location and intended destination, and to get simple directions, and a rough time-estimate. All programmed in DFns (and very nicely described by John Scholes at the May Vendor Forum) with a simple but fully-functional Gui.

They even had an early version of *Rain* running, complete with as much of the viewer as fitted on the post-card-sized screen! In fact almost any reasonable Dyalog application should at least have a go, straight out of the box, as long as it keeps away from Windows API calls and is sensible with the Gui.

So, Adrian looked at Jonathan and Jonathan looked at Adrian and we said "Want one!", so after a little research we have a matched pair of HP Jornada 568 clamshells, called CEcil and CElia respectively. Now comes the hard part ...

Can We Make Them Do Something Useful?

Oddly enough, I already had a target application in mind, which did help to get the authorisation past the APL385 purchasing department. Every few months the Ryedale Field Naturalists (see www.ryenats.org.uk) go out in search of birds or flowers or fungi. Having spent an exciting day peering into dense thickets or sinking into bogs, they return triumphant with a long list of the botany and assorted avifauna they have sighted. In former times, the list consisted of scribbled notes on a clipboard, which some muggins then had to reformat into a neat CSV file so it could be published on the website for all to admire.

Typically, the list will be over 100 flowers in a day's outing, so you can guess that transforming the hurried notes into an accurate list with both Common and Latin names is a tedious and error-prone business. As it happens, GraPL is a handy tool for reading in a CSV and writing out a nice HTML table, so all we need is something we can take out in the field which will speed up the process of creating the CSV in the first place.

Of course, one could simply use CEcil and either Excel or Notepad to enter the data by hand, but I have accompanied enough of these trips to know that the quickfire "Look, here's a banded Knotweed with asymmetric follicles ... it must be *Albinensis rusticana*" (sorry I made that one up) would swamp any attempt to work on a normal keyboard, let alone a tap-and-shoot interface. The key to success would surely be to ensure the absolute minimum of typing, and yet guarantee total accuracy in the spelling of the final result. Spurred on by the memory of POET (Portable Order Entry Terminal – Rowntrees circa 1980) and the 4K of Marconi Assembler that my colleague Dr Wilson once wrote, I set to work on the design.

A vital resource which we already had was the BSBI official list of all the plants known to grow in the UK, listed by "BSBI number", Latin name and Common name. Of course there were far too many of these to contemplate the idea of a drop-down list (around 6,000 I think) and anyway, how would you arrange the drop-down. Sometimes it is a Latin name that is shouted across the field, sometimes the common name, sometime a mixture of both ("Ranunculus, Creeping I think ..."). The first rule of *Extreme Programming* is "Build the simplest thing that could possibly work" which in this case was going to be an edit field where you could enter something like "ra cre" and tap "Find" to get a list of all the likely candidates. Maybe there would only be one, or maybe several. Then the intrepid botanist could simply tap the correct one and hit "Add to cart" to log it. It had to work simply and quickly, but of course it did need to run some obvious checks, such as making sure you only logged each plant once.

So, here we are with our Dyalog 10 engine, our BSBI list and some vague ideas about design. It's Saturday and the Terrington trip is on Tuesday. Time to stop brooding and start cutting code.

Some Early Thoughts on the Gui

As I'm sure you know, a *Causeway* is a safe path across a bog, so a good start would be a port of just enough of *CPro* (let's call it CProCE) to get me going. Having spent a few days just playing, I had come to the view that the 'typical' Pocket application used only the most basic Gui controls. No trees, no listviews, no fancy buttonbars. Remarkably Win31 in fact, so let's start with the classes marked 'Win31' in standard Causeway and see how far we get. A quick reading of the help file showed a few critical changes that would have to be made to the 'Form' class so it would behave correctly (for example modal forms would need to close and return 1 when the user hit the OK in the corner) and of course the default sizes for edit fields and buttons were all a little different. However this is mostly in data (the `Design` variable in the class namespace) so getting it all up and running took only a couple of hours.

Rather than porting the *CPro* Designer, I simply set the form size to match the Pocket display, and set the default font to MS Sans Serif Bold so when the form was created on the 'real' PC it was a close match for the actual appearance on the Pocket machine. That way, all I needed was a global switch to check the Dyalog version and enable the special behaviours which only the Pocket APL supported.

Now I could design and test the entire application in relative comfort, and simply `)save` it to a subfolder that ActiveSynch was looking out for. Wait a couple of seconds until the logo on the system tray want from amber to green, reload the workspace on CEcil and try it for real. Swear at an un-noticed *⎕NA* call, fix or remove it, resave, count to 10, reload and try again. It sounds messy, but actually it works fine – a bit like having a dual monitor layout apart from the short delay while the WS is copied down the USB link.

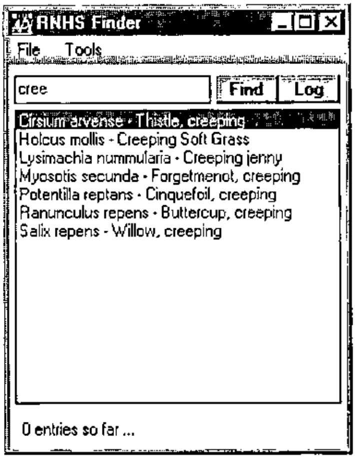
Teething troubles

All these got fixed really quickly, but I should probably mention them in passing. There was a very strange effect caused by a line in the *CPro* engine which ran *⎕DR*¨ on a column of a nested matrix (by the time Dyadic had fixed the bug I had removed the offending code); native files did not work at all (so I had to save the plant-list in a component file and write it to CSV with 'real' Dyalog-9); the main form did not minimise properly when its X button was hit, so CEcil was a one-application machine for the day. Sometimes the "context menu" event seemed to go missing, but I only made minimal use of tap-and-hold menus, so not a big problem.

The good news is that the *CPro* namespace is now completely common to the PC and CE platforms. All we need to do is plug in a different (and much smaller) class library. The *Designer* also needed no changes at all, as long as you only develop on the PC. To make a fully usable version of this for the Pocket display would be a lot of work, but I may well make a minimal version which at least allows me to debug forms and fix the "Data to watch" and "Behaviour" entries over on the Pocket. I doubt if I will ever do the visual design stuff "over there" as I cannot really see the need. Other opinions welcome, of course.

Building and Installing the Finished Application

As you might guess, the main form only took a few minutes to rough out, and looks like this:



cree

Cirsium arvense - Thistle, creeping
Holcus mollis - Creeping Soft Grass
Lysimachia nummularia - Creeping jenny
Myosotis secunda - Forgetmenot, creeping
Potentilla reptans - Cinquefoil, creeping
Ranunculus repens - Buttercup, creeping
Salix repens - Willow, creeping

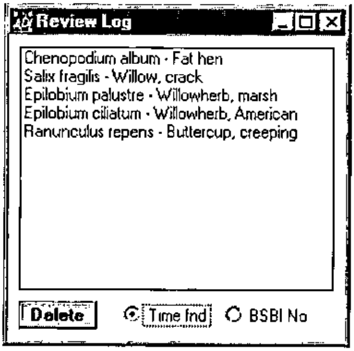
0 entries so far ...

I now needed to put some code behind the 'Find' button, and speed was likely to be an issue, so I pre-processed as much as I could to make the text search as fast as possible. The first thing I do is save the full namelist in uppercase to a cr-delimited simple vector with blanks added to the front and back of each line. I have a matching numeric vector to tell me which row each line comes from, so all I need to do is use the absurdly fast ⍷ to match each part-word the user typed and progressively eliminate candidate indices. Here is the crucial code:

```
  ids←Find plants;plant;txt;hits;ptn
⍝ Check the full list of names ∆nm for hits on the passed <plants>
⍝ Each word is taken in turn so 'but cree' would hit 'Creeping Buttercup'
⍝ Returns the ids from ∆idx of all the candidates
⍝ See Init which makes the searchable text

 plants←#.util.toupper plants
 ((plants∊'-')/plants)←' '
 plants←plants~''''
 plants←' ',¨' ,;'#.util.∆csvton plants

 :If 0∊⍴plants
   ids←⍬
   :Return
 :End

⍝ Mask hits from each word
 ids←∆id
 :For plant :In plants
   hits←(plant⍷∆txt)/∆ptn
   ids∩←∆id[hits]
 :End
```

On the HP Jornada (ARM chip) this runs comfortably quick enough (under 1/10th of a second) on the 27571-byte text vector, so no problems keeping up with my handwriting speed!

Of course users make mistakes, and botanists were likely to ask "How many have we got so far?" so the other screen I needed was to allow deletion from the log, as well as to show a simple count of the number of entries so far. One of the things you cannot avoid on the Pocket Gui is a menubar, so let's use it here! "Tools" seems a good place to bury this one, so we have something like:



Chenopodium album - Fat hen
Salix fragilis - Willow, crack
Epilobium palustre - Willowherb, marsh
Epilobium ciliatum - Willowherb, American
Ranunculus repens - Buttercup, creeping

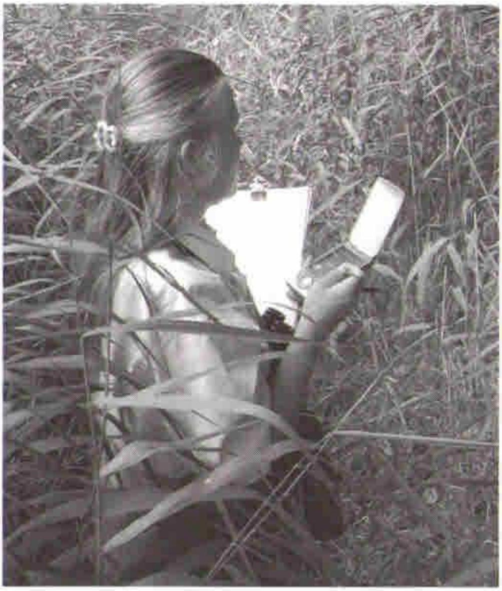
Simple enough, and this one is modal so it really goes away when you hit the X in the corner. A few fragments of utility code (like Save and Revert) and we are ready to give it a tryout.

(The radio buttons choose the ordering – a very late-running field upgrade!)

Implementation issues

Nothing to do with Dyadic, but something to be aware of. It is *amazingly* hard to do the basic maintenance tasks with the CE interface. Yes, it is Windows, but it does its damndest to pretend that it is just another dumb organiser! There really is no way to edit the properties of a shortcut, let alone add a new one, with the supplied interface. As for changing stuff in the Registry, don't even think about it!

Fortunately, there seem to be loads of useful tools out there to fill the gaps, and most of them are free. A quick trawl on the internet turned up *GSFinder* as a 'proper' Explorer (actually rather a good one) and pocket *RegEdit* and *Notepad* soon followed. Now it is possible to make a new shortcut to start Dyalog, load my workspace and in the usual way have ⎕*LX* run up the main form. Of course we do not yet have a 'runtime' environment, but I can check to see if I am running on the Pocket version of APL when the form closes and ⎕*OFF* if so. If I add the shortcut to the Start menu (one of only 9 allowed) then we have something that has all the look and feel of a real shippable application.



As you can see, it really did work. Next time we are going to fix up some Velcro so CEcil is actually stuck to the clipboard, but in general the ergonomics were very satisfactory. I was tending to use the 'hunt and tap' keyboard, where Gill was happier with the letter-recogniser. However both were accepted with no problems by the standard text-input field so it was easy to choose. One major benefit of the HP screen was that it was comfortably readable out of doors in full sunlight.

This, coupled with the remarkable battery life (I took it to Italy for a 5-day trip, used it as my alarm clock every morning, and was still writing code on the train home) makes it a very handy tool for use out in the field, or even the jungle!

Can we Develop APL Code with it?

The only way to answer this question was to give it a try. I really can't see myself writing any serious Gui applications in this environment, but what about those little nuggets of code you need to put behind the Gui? The sort of thing that you worry away at while waiting for planes and trains, in fact. On the way back from Milan I thought it would be good to cut some real code, and hopefully make a good set of test examples to prove it.

The problem I had was simple enough to specify: given a simple numeric matrix, take an index vector (typically 2 numbers) which could be integer or floating-point and return a scalar value. Integer indices do what you would expect, floating-point values result in a simple linear interpolation on the corresponding axis. We do not attempt to extrapolate over the ends.

Simple enough, but a few cases to test and enough code to write to give the APL keyboard a reasonable work-out. How easy was it to get this working?

Using the session

Obviously you need this to generate your test data, select some functions to edit, and then to run. I soon found I was setting ⎕*PW* to be fairly wide, as otherwise even quite small matrices scrolled the session much too far for comfort. Using the rocker-pad to scroll up is fine for 2 or 3 lines, but not for much more than this. I wonder about using the old VSPC style session here – separate area for input at the foot of the screen and a special command to 'bring down' output lines into it.

I also found working with multiple windows very cumbersome, and quickly gave up attempts to fix code 'on the fly' in the editor. It seemed much less confusing just to look at a few variables, hit → and start again from the top. It never tried the tracer at all, but then again I never use it anyway, prefering the old ∘ approach to stopping code in its tracks. I think Dyadic should take a hard look at the APL2000 *Code-walker* which would be a far more appropriate style of interface in the limited screen-space they have available here.

Working in the editor

Actually typing in the APL code was the least of the problems. The auto-complete is excellent, as it can guess from the context which names are visible and usually homes in almost instantly. It almost makes the idea of putting the control-structures on one of the sets of virtual keytops redundant, as they auto-complete very quickly from the normal Alpha keypad. Of course typing the symbols is no harder than typing anything else, although I continue to wonder about the logic of perpetuating an association of symbols with keytops on a machine which does not actually have any keytops! Something more like the Word Equation-editor toolbar might have been more appropriate here? *Edit, Format* is a godsend here, as lining up code by hand is not something you would want to do.

Lessons for Mainstream Interpreters

Two things are simple and obvious.

1. The auto-complete is really useful when you are digging around in a tree of namespaces. The fact that it can restrict the search to the likely valid names means that it offers useful guesses with a minimum of typing.

2. We really must start offering novice APLers some keyboard help. Never mind showing a map of the keyboard – please can we just have an optional toolbar with the symbols on it, sensibly grouped. If APL2C can do it, why can't the rest of you? I suspect even experienced APLers would find it helpful from time to time. I still type ,[1] rather than {commabar} in APL+Win because it is faster and a lot less error-prone than finding the symbol!

I think the final lesson is a bit harder to implement. I agree with Ray that we can actually generate useful applications for this little gem of a machine, and that at the moment the market is wide open. However I really think that we want a much lighter-weight installation, and that Dyalog may finally have to follow +Win and APLX in making the 'runtime' interpreter just that, rather than simply the development system with a flag set! Remember that this beast has no hard-drive, so any install is taking a fair chunk out of the 40Mb of free installed memory. Hopefully, this message will get through to the developers, and we can also benefit from a lighter-weight installation for the 'real' PC platform as a result. If we can all share the same interpreter, so much the better.

Summary

This is a computer, and an APL implementation, that you should take seriously. It can do real work, and it really can go out in the field and do useful stuff under 'battle' conditions. As a machine to take with you on that 10-hour flight, it is hard to beat.

APL 2002 AT MADRID

reported by Anthony Camacho

APL 2002 was held from 22 to 25 July at the Universidad Autonoma de Madrid.

It was well organised. The display equipment worked. The rooms used were never overcrowded. There was an excellent computer workshop with internet access. The food was acceptable. The proceedings were given out at registration – a significant benefit as it enabled delegates to make a much better informed choice of papers to attend. Not only did every delegate get a Vector and a British APL Association CD containing the proceedings, the Vector archive and some free APLs and some of Adrian's pictures, but also there were copies for everyone of *Les Nouvelles D'APL* from AFAPL and also another CD with A+ in versions for Windows and Unix from Morgan Stanley. There was an excellent conference bag, well designed, well made and with numerous zips and pockets, but there were no T-shirts or other give-aways. (The BAA CD is to be sent to all subscribers to Vector with issue 19.2.)

There were few hitches and they were all well dealt with; the worst was that on the first morning the coach to take us from the hotel to the university did not appear and we were carried in taxis after only about twenty minutes delay and the conference still started almost on time.

There were some good professional presentations, in particular from Dyadic, MicroAPL, Adrian Smith, APL 2000 and Soliton. The best presented conference paper was by Alan Mayer and Alan Sykes. There were some presentations where the author had done nothing to prepare except write the paper and/or had not tried a run through; you know who you are so we won't add to your embarrassment by naming you.

The University is about half an hour from the centre of Madrid and is quite new. The building where the conference was held was opened in 1999 and felt a bit bleak. There was no place with comfortable chairs to relax and talk if no paper appealed, except the computer room, and since the buses left the hotels at eight thirty or so and did not get back till seven or after, the days were long and tiring.

There are 91 names on the list of participants; as far as I am aware only one paper had to be cancelled because there was no presenter for it.

The first sighting of Dyalog for the PocketPC
John Daintree demonstrates the London Routefinder

I suspect the highlight of the conference was when John Daintree showed a pocket PC with a full implementation of **Dyalog APL** - an excellent additional reason for keeping your functions (and function lines) short! I want one.

APL 2000 had significant enhancements to announce, **MicroAPL** showed their excellent new multi-platform APL and Dyadic introduced us to the joys of .NET.

At the banquet the only speech was by Bob Brown who explained that the venue for APL 2003 was not yet decided and that the SigAPL committee had not yet agreed on a recipient of the SigAPL outstanding achievement award for this year. Lynne Shaw presented Jon McGrew with the plaque for last year's award, which was not ready in time to give it to him at Yale (APL 2001). Everyone toasted Manuel Alfonseca (the Chairman of the conference) and his staff for an excellently organised and run conference. He deserves thanks from all of us.

My Impressions of the Conference

from Ian Clark

I for one did enjoy the conference, and since I paid my own way from start to finish (that must be a first!) I felt I'd got my money's worth.

But no thanks to the formal content, which was pretty dismal. I did go to the last day, and I found the open discussion unexpectedly useful. I wish I'd taken notes. The rest wasn't up to much.

Apart from the vendor material, always worthwhile, if only to see what they're **not** doing, the only two papers that thrilled me were the one by the Alans and, surprisingly, Paul Cockshott's on Vector Pascal. I'd reviewed that one, and warned him away from making too 'academic' a paper out of it as he was proposing to. I thought it was a long-shot for this conference, but if there was a theme for me, it was how to extend existing procedural languages to give APL-like array functionality, and Vector Pascal was right in there. Its application domain (effectively the cinema of the future) was fascinating to me.

Jonathan Manktelow with Paul Cockshott

So where was the problem? Couldn't fault the organisation – although anything university-based has to reconcile the disparate needs of good hotel and food versus cheap university accommodation, which can mean a long trip between comfortable surroundings to chat and the actual presentations. As in this case. Too far to go back for a hanky. Like Anthony, I feel that if everything isn't on-site it makes a long day of it. Particularly if there's nowhere to chill-out—nowhere you don't get shooed out of.

I seem to recall some discussion in the open session on Thursday of commercially-based versus academically-based APL conferences. I don't think the APL world is dead by any means (last November at Naples, Florida showed a lot of life) but it may be in danger of fragmentation. I think there are two issues. The comfort issue noted above, which TU-Berlin solved by being a city-centre university site close to the main hotels, and having excellent refreshment and sitting-around facilities just outside the (albeit severe) teaching rooms. And the content issue.

For the sake of clarity I'm going to be beastly and cruel here. I get the feeling with Hispanics that they think that if they just ignore the English-speaking world it will eventually go away. Yet here was Manuel hoping to help his university come out of the Franco closet into the Euro scene, indeed the world scene as represented by APL (which he might hope to be less English-dominated than most computer topics) and yet all it did was expose Spain for the backwater it still is. If we choose in future to have the world conference in a minor-league country, APL-wise (I'd be very careful to exclude say Finland or Denmark from such a designation!) then it's going to be most important to act as uncle or guardian angel to the local group, without stealing its thunder, to ensure that major players get to submit good papers, and that the schedule doesn't get filled with who-he's who are desperate for the shop-window. If a good enough stream of papers don't materialise, then a bit of cajoling and bullying will need to be done. And not just to plead for fillers from people who are going to turn up and simply ad-lib, or read out their vanilla-flavoured papers word-by-word, but by people who can be guaranteed to put on a good show.

Inner Product Fractals from Fuzzy Logics

by Angela Coxe (coxea@lafayette.edu)

Introduction

Fractals have become a popular topic to study because of their aesthetic appeal. They have been used in different fields and also as a way to depict natural landscapes [2,7]. One classic fractal is the Sierpinski Triangle [5]. This fractal may be constructed using J inner products [3]. In this note our goal is to explore the use of fuzzy logics with such inner products. This grew out of a project I did in a visualization class [4].

Boolean Inner Product Fractals

We first consider using an inner product in J to construct the Sierpinski Triangle. To make the images we start with the binary representation of the values. Here is a matrix of 3-digit and base-2 construction of binary representations.

```
   k=: 3
   s=: 2
   ]m=: (k#s)#: i.s^k
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

Then we take the "or" of pair-wise "and" of each row of *m* with each column of the transpose using a J inner product; see the following. The 0's correspond to the Sierpinski Triangle.

```
   ]b=: m +./ . *. |: m
0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 1 1 1 0 1 1 1
0 0 0 0 1 1 1 1
0 1 0 1 1 1 1 1
0 0 1 1 1 1 1 1
0 1 1 1 1 1 1 1
```

We then increased the value of k to 8 and made the picture in Figure 1. By changing the "or" and "and" operators (`+.` and `*.`), values of the digit, and the base in the above expression, different variations on the Sierpinski Triangle can be seen. The inner product matrix above with a 5-digit base-3 representation gives the Sierpinski Carpet seen in Figure 2, where `+.` and `*.` were the gcd and lcm.

Fuzzy Logic

Fuzzy logic is a way to quantify indefinite values of truth. There are different types of fuzzy logics and each type gives a different behaviour. Fuzzy logics use "or" and "and" operators, and variations of "or" and "and" to determine the differing types of fuzzy logics. The types we are going to examine first are the probabilistic fuzzy logic and the max/min fuzzy logic.

We start with by showing the Boolean Logic truth tables in Tables 1 and 2.

	0	1
0	0	1
1	1	1

Table 1. Boolean "or" truth table

	0	1
0	0	0
1	0	1

Table 2. Boolean "and" truth table

In the probabilistic fuzzy logic x or y is $(x + y)$-$(x * y)$ while x and y is $x * y$ [4,6]. Here is the implementation in J.

```
por=: (+ - *) 0
pand=: *
```

Tables 3 and 4 are the truth tables for the probabilistic logic. Comparing these tables to the Boolean truth tables (Tables 1 and 2), note the 0's and 1's are positioned as in the Boolean tables.

	0	0.25	0.5	0.75	1
0	0	0.25	0.5	0.75	1
0.25	0.25	0.4375	0.625	0.8125	1
0.5	0.5	0.625	0.75	0.875	1
0.75	0.75	0.8125	0.875	0.9375	1
1	1	1	1	1	1

Table 3. Probabilistic "or" truth table

	0	0.25	0.5	0.75	1
0	0	0	0	0	0
0.25	0	0.625	0.125	0.1875	0.25
0.5	0	0.125	0.25	0.375	0.5
0.75	0	0.1875	0.375	0.5625	0.75
1	0	0.25	0.5	0.75	1

Table 4. Probabilistic "and" truth table

Probabilistic Fuzzy Logic Inner Product Fractal

We will now look at the probabilistic fuzzy logic and apply it to inner product fractals. The expression below makes a matrix using *k*-digits and *s*-sampled values.

```
  k=: 2
    s=: 3
    ]m=: >,/{k#<(i.%<:)s
  0   0
  0 0.5
  0   1
0.5   0
0.5 0.5
0.5   1
  1   0
  1 0.5
  1   1
```

Fuzzy logics may be used on this *m* in the same construction as the inner product fractals discussed earlier with a slight variation to accommodate the fuzzy "or" and "and" operators.

```
    ]b=: m por/ . pand |:m
0    0   0    0      0     0   0     0    0
0 0.25 0.5    0   0.25   0.5   0  0.25  0.5
0  0.5   1    0    0.5     1   0   0.5    1
0    0   0 0.25   0.25  0.25 0.5   0.5  0.5
0 0.25 0.5 0.25 0.4375 0.625 0.5 0.625 0.75
0  0.5   1 0.25  0.625     1 0.5  0.75    1
0    0   0  0.5    0.5   0.5   1     1    1
0 0.25 0.5  0.5  0.625  0.75   1     1    1
0  0.5   1  0.5   0.75     1   1     1    1
```

In order to visualize a more detailed version, we assign each value in our matrix to an integer between 0 and 255. The linear visualizing is done using the function `lin256`, which matches integers to a palette to construct colour pictures [4]. Notice how there is a block of 1's in the lower right corner. The 1's correspond to the color black in the palette, the 0's correspond to the color white in the palette, and the rest of the values are different hues in between. Here is the J representation used to create pictures using the probabilistic fuzzy logic.

```
   load 'fvj2/raster5.ijs'
   lin256=: <.@(255.99&*)

   $m=: >,/^:(_1: + #@$){3#<(i.%<:)8
512 3
   $b=: m por / . pand |: m
512 512
   (P256; lin256 b) writebmp8 'pFuzzy000.bmp'
```

See Figure 3 for an example of inner product fractals using the probabilistic fuzzy logic. This figure shows the nesting with shifts of shading within the image. Recall that the lower right side is a black block because all the values in that portion of the matrix are 1's.

Max/Min Fuzzy Logic

Another type of fuzzy logic is max/min. This logic uses max (>.) and min (<.) functions. The following is the implementation of the max/min "or" and "and" functions.

```
mor=: >.
mand=: <.
```

Tables 5 and 6 show the truth tables for the max/min fuzzy logic.

	0	0.25	0.5	0.75	1
0	0	0.25	0.5	0.75	1
0.25	0.25	0.25	0.5	0.75	1
0.5	0.5	0.5	0.5	0.75	1
0.75	0.75	0.75	0.75	0.75	1
1	1	1	1	1	1

Table 5. Max/Min "or" truth table

	0	0.25	0.5	0.75	1
0	0	0	0	0	0
0.25	0	0.25	0.25	0.25	0.25
0.5	0	0.25	0.5	0.5	0.5
0.75	0	0.25	0.5	0.75	0.75
1	0	0.25	0.5	0.75	1

Table 6. Max/Min "and" truth table

We can see the values in the max/min truth tables all appear in the domain, while the probabilistic truth tables contain additional values outside the domain.

As before, this is substituted into the inner product formula and the rest is the same as the probabilistic logic.

```
    $m=: >,/{3#<(i.%<:)8
512 3
    $b=: m max / . min |: m
512 512
    (P256; lin256 b) writebmp8 'mfuzzy000.bmp'
```

An illustration of max/min fuzzy logic inner product fractal is seen in Figure 4. Notice the uniform bands of color within the nesting of shades. This is not surprising given the patterns in Tables 5 and 6.

Schweizer Parameterized Family of Fuzzy Logic

Next, we are using the Schweizer parameterized family of fuzzy logic to create a family of inner product fractals. For this type we are using two definitions, which give the fuzzy logic for any $\alpha>0$ [6].

The Schweizer "or" definition is:

$$x \text{ sor } y = (1/x^{\alpha} + 1/y^{\alpha} -1)^{-1/\alpha}$$

and the implementation in J is:

```
sor=: 1 : 0"0
:
if. 1= x.>.y. do. 1 else.
1 - ((%((1-x.)^m.)) + (%((1-y.)^m.)) -1)^(-%m.)
end.
)
```

The Schweizer "and" definition is:

$$x \text{ sand } y = 1 - (1/(1-x)^{\alpha} + 1/(1-y)^{\alpha} -1)^{-1/\alpha}$$

and the implementation in J is:

```
sand=: 1 : 0"0
:
if. 0 = x.*y. do. 0 else.
((%(x.)^m.) + (%(y.)^m.) - 1)^(-%m.)
end.
)
```

The J adverbs for Schweizer families are run similar to the above fuzzy logics. The difference in the J implementation is the α-value placed in front of the `sor` and `sand` resulting in the appropriate function. Each different α-values affects the picture slightly. Here is the variation of the inner product fractal used with the α-value of 0.5.

```
   $m=: >,/^:(_1: + #@$){3#<(i.%<:)8
512 3
   $b=: m 2 sor / . (2 sand) |: m
512 512
   (P256; lin256 b) writebmp8 'sfuzzy000.bmp'
```

The Schweizer image with the α-value of 0.01 behaves virtually identical to Figure 3. In addition, Figures 4 and the Schweizer image with the α-value of 20 also have virtually identical behaviours. The images I produced with the Schweizer family showed an α-value greater than 1 had behaviour similar to a max/min fuzzy logic, and an α-value approaching 0 had behaviour similar to the probabilistic fuzzy logic. Figure 5 is an example where the α-value is between the probabilistic and max/min figures. To see an animation of this Schweizer family see [1]. A beta version of image3 addon for J was used to create the animation. This page also another example of the probabilistic logic were the argument is preprocessed by the following function.

$$tw(x)= \frac{1}{2}+m(x-\frac{1}{2})+4(1-m)(x-\frac{1}{2})^3$$

We call this the twisted family of logic, and Figure 6 contains an example of the twisted logic at the initial value of –3.

Conclusion

We have seen how fuzzy logics in inner product constructions form fractal like images. Parameterized families of fuzzy logic show how variations on the logics according to the parameters give smooth variations between images.

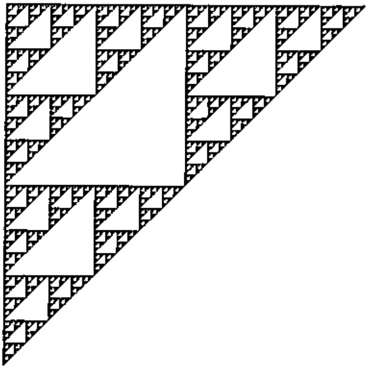


Figure 1. Sierpinski Triangle from inner product fractal of 8-digit base-2 representations.

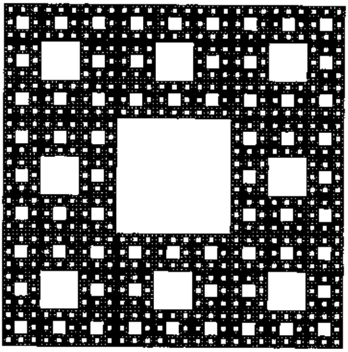


Figure 2. Sierpinski Carpet from a base-3 construction

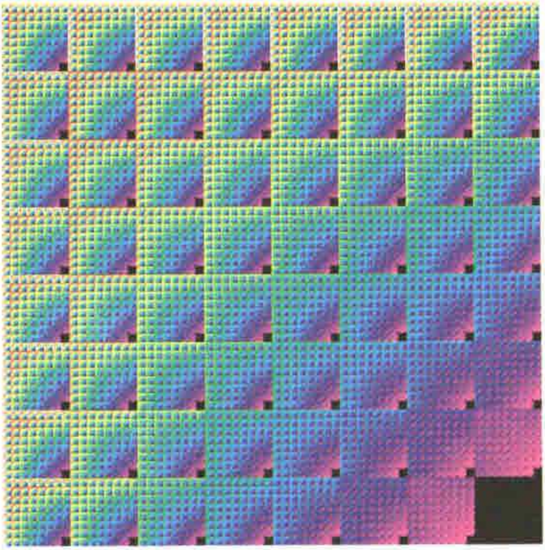


Figure 3. **Inner product fractal from 3-digit probabilistic fuzzy logic sampled at 8-points.**

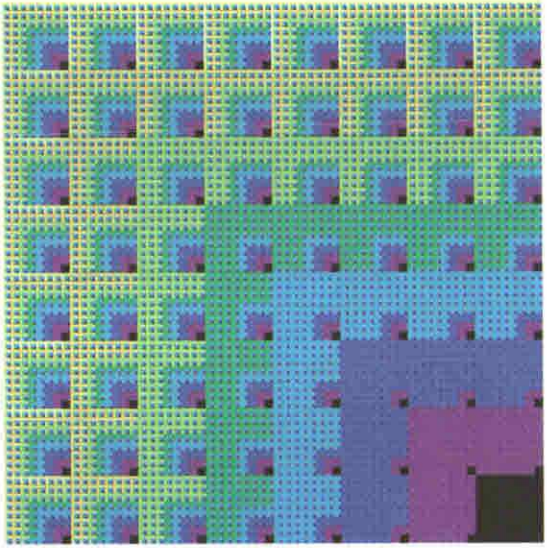


Figure 4. **Inner product fractal from 3-digit max/min fuzzy logic sampled at 8-points.**

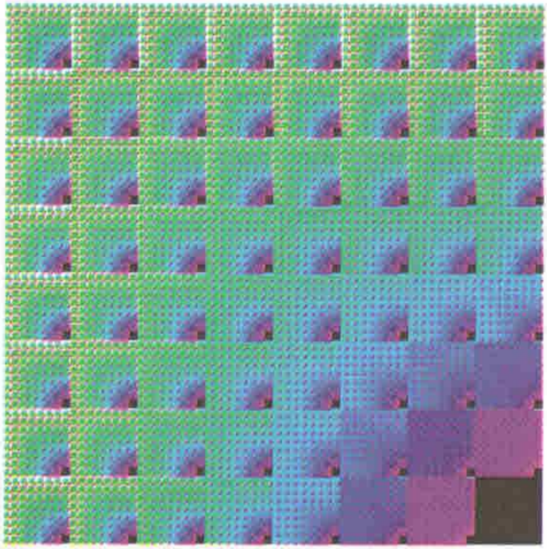


Figure 5. Inner product fractal from 3-digit Schweizer fuzzy logic sampled at 8-points with an α-value of 2.

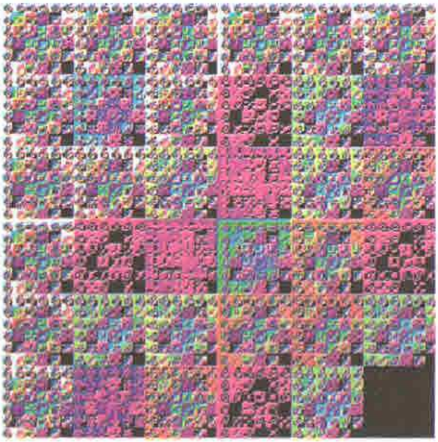


Figure 6. Twisted inner product fractal with a parameter value of –3.

References

[1] Angela M. Coxe, Auxillary Materials for "Inner Product Fractals from Fuzzy Logics", http://www.lafayette.edu/~reiterc/sp02fp/coxe/index.html

[2] Barnsley, Michael. *Fractals Everywhere*. Academic Press, Inc., San Diega, CA (1988).

[3] C. A. Reiter, Fractals and Generalized Inner Products. *Chaos, Solutions and Fractals* 3, 695-713 (1993).

[4] C.A. Reiter, *Fractals Visualization and J 2nd Ed*. Jsoftware, Toronto, Canada (2000).

[5] C.A. Reiter, Sierpinski Fractals and GCDs. *Computers & Graphics*, (18) 6, 885-891 (1994).

[6] H.T. Nguyen, E.A. Walker. *A First Course in Fuzzy Logic, Second Ed*. Chapman & Hall/CRC, Boca Raton, FL (2000).

[7] Mandelbrot, Benoit B. *The Fractal Geometry of Nature*. W.H.Freeman and Company, New York, NY (1983).

Angela M. Coxe
Lafayette College Box 8916
Easton, PA 18042, USA
coxea@lafayette.edu

A Gui-quad for Graphical User Interaction

by F.H.D.(Eke) van Batenburg [1] [2]
Jasper van Dijk [2], *Dirk-Jan van Leeuwen* [2]

Abstract

This paper demonstrates a gui-quad primitive that should be added to APL. This gui-quad is a logical successor of quad and quoted-quad for a windowing environment.

The right argument of gui-quad is a vector with data-items that are to be presented to the end-user and the result is the same vector with these data items modified as the user saw fit. The left argument specifies the layout. So the right argument specifies what should be presented and the left argument specifies how.

The functional repertoire of this gui-quad is not so extensive that it can supplant the ⎕*WI* primitive. However, this gui-quad primitive is far easier to use and also satisfies most of the programmer needs for a gui-like interaction with the user.

Introduction

A good thing in my opinion is that most APL vendors have, despite some undeniable differences, an APL that is very similar with regard to the primitive functions, syntax and basic primitive variables. For this reason when I switch from my PC where I use APL2C or APL2000 to the Mac where I use APL68000 (now APLX) most of my coding still works. And even when I would move to Unix, a lot (I refrain from saying most) of my APL coding still works in SAX APL. And this is more or less codified in the newest ISO standard [1]. You could make an argument as to whether it was the ISO standard that standardized APL, or whether the vendors adhered to common principles that made an ISO standard possible or a bit of both, but I do not intend to raise that discussion here. The fact remains that I can work without too much mental juggling in many different APLs.

[1] Batenburg@rulsfb.LeidenUniv.nl

[2] Theoretical Biology, Inst.Ecological & Evolutionary studies, University Leiden, The Netherlands

The thing that is not codified in the ISO standard is the graphical user interface in APL. And in the beginning this used to be very different in different APL implementations. Despite the initial differences I noticed that this initially wide variety of approaches seemed to have grown towards each other in the last decade. The first one with a nice graphical user interface was APL68000 before the DOS-world even heard of the words graphical user interface. Then APL*PLUS came with a graphical user interface using ⎕WGET and ⎕WPUT that was very nice taking into account the primitive DOS environment of that time. When Dyalog introduced their ⎕WI using an object-oriented programming approach they initiated a new and powerful gui for APL. First APL*PLUS adopted this (I leave it to the reader to pronounce APL*PLUS as APL/Manugistics, Cognos, APL/Win or APL2000) and now APL68000 (today to be pronounced as APLX of MicroAPL) adopted it too. This "organic" standardization is something that I am very happy with.

User Interface.

Having said that I am happy with the standardization of ⎕WI, I must add that as a programmer I have mixed feelings about the ⎕WI. On the one hand it is an extremely powerful tool for a programmer. On the other hand unfortunately it is indeed true that ⎕WI means "Window Interface". My point is that as a programmer, I do not want to interface with Windows (imagine how my students laugh when I work on my Mac and I explain to them that I am interfacing with "Windows"). I do not want to interface with Windows, I want to interact with the User.

Now APL used to have perfectly good primitives to interact with the user: the quad and the quote quad. It is clear that in the current operating systems the quad and the quote quad are inadequate for a modern user interface. But rather than a Windows Interface I think there is a need for a "Graphical-User-Interaction-Quad", in short a gui-quad.

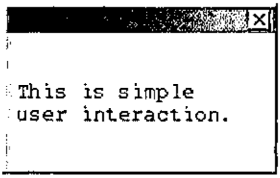
Such a gui-quad should take the information that the programmer wants to present to the end-user, show it in a nice way, let the user change it as he or she sees fit, and return the modified result. For this I do not want to "interact with Windows", I do not want to make forms, to create objects nor do I want to assign properties.

Basic interaction

So let me be more concrete in what I do want, in other words how I want the gui-quad to work. In its simplest form I just want to say something like:

```
⌻ ⊃'This is simple' 'user interaction.'          [3]
```

and I expect to produce something like:



```
This is simple
user interaction.
```

```
0 ⍝ the explicit result that will be explained later
```

You see? No forms, no ⎕WI, no ⎕WCALL 'MessageBox', no more programming overhead than just that one symbol for graphical user interfacing.

Now as you see this is a very basic thing. We only have to supply the information that we want to present to the user. Just like in the old days when we said:

```
⎕ ← ⊃'This is simple' 'user interaction.'
```

If we want to enhance the layout we should give slightly more information to υ . For example we could specify that we would like to display this information in a query-box, or a proper info-box. For this we supply layout information as a left argument. For example:

```
'?'⌻⊃'This displays in' 'a query box.'
'i'⌻⊃'This displays in' 'an info box.'
```

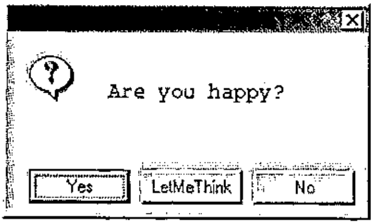
For real interaction you want information back from the user. The simplest way would be to enable him to answer by clicking on appropriate exit buttons. Of course you do not want to be restricted to Yes/No or OK/Cancel. What about specifying these options at the left? Something like:

```
⎕ ← '?' 'Yes' 'LetMeThink' 'No'⌻ 'Are you happy?'
```

which generates:

[3] Personally I would have preferred to use the symbol ⍈, but unfortunately this is already used for component files in APLX. So instead I selected the symbol ⌻, where the ⎕ suggests input/output and the "∘" inside suggests the gui object "option" or "radio-button".

and returns:



Are you happy?

```
  2
```

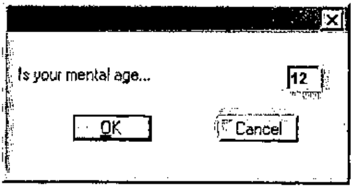
Now we have a more solid interaction. I presented a question, and the user reacted by clicking one of the exit buttons. The result was a number; 1 for [Yes], 2 for [LetMeThink], 3 for [No] and 0 if the user clicked the top-right button [X]; you see that the user clicked [LetMeThink] because the result was 2.

Summarizing, I would like a "graphical-user-interaction-quad" that does not require me to specify all kinds of Window objects, but permits me to supply my information to its right (and if necessary some minimal layout specification at its left) and which returns to me the user response.

More subtle user interaction

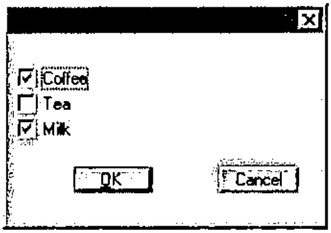
The previous example, although rather basic, illustrated the basic principle. Now a graphical user interaction needs subtler interaction than by simple exit-buttons. Often I have some (default) data that I want to present to the user and give him the opportunity to change those parts that he disagrees with. And I do want to present the data in a gui-type way, for example as radiobuttons, clickboxes, number or text fields, or sliders. Again, we would like gui-quad to take the information to its right, get layout specifics at its left, and return the presented information back with modifications if the user changed it. For example:

```
  ⎕ ← '#'⌸ 'Is your mental age...' 12  ⍝ number field
```



Is your mental age... 12

```
  1 64 ⍝ the user disagreed with 12 and entered 64

  ⎕ ← 'x'⌹ 'Coffee' 1 'Tea' 0 'Milk' 1 ⍝ click boxes
```



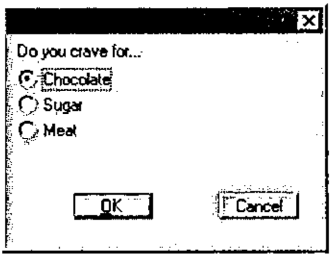
☑ Coffee
□ Tea
☑ Milk

```
  1 0 1 0 ⍝ the user only liked tea
```

This example showed that a more complex question may need a general introductory text. For this we supply a nested array with two elements, the first one with the general text and the second element with the detailed information. For example:

```
  ⎕ ← 'o'⌹'Do you crave for...'('Chocolate'1'Sugar'0'Meat'0)
```

The left argument 'o' is a radio-button specification so we get:

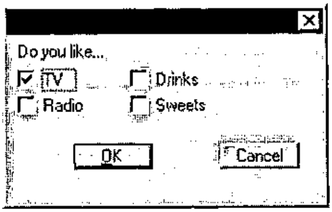


Do you crave for...
● Chocolate
○ Sugar
○ Meat

```
  1 2 ⍝ the user was a sugar lover
```

All the previous examples only queried the user for one single set of alternatives. Would it not be nice if we could ask the user several sets of questions? As you see in the examples each basic unit is a text and a value. Upto now, we used several of such pairs in a vector, but we can also ask the user his or her opinion on several sets of alternative pairs. Instead of a vector with questions (which is handled similarly to a double column of text-value-pairs) we can offer a matrix with several column-pairs. For example:

```
  ⎕ ← 'x'⌹'Do you like...'(2 4⍴'TV'1'Drinks'0'Radio'0'Sweets'0)
```



Do you like...

☑ TV □ Drinks

□ Radio □ Sweets

```
  1 1 0 0 1
```

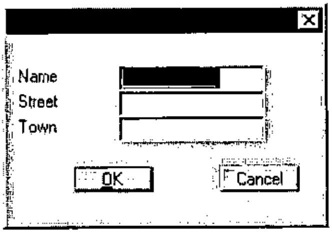
The result is a vector with first (as always) a number that indicates how the dialogue ended (1 for [OK], 2 for [Cancel] and 0 for [X]), followed by the user response on each of the four clickboxes.

These examples sufficiently show how you can present numbers to the user for modification with a layout specification at the left.

Text is very similar to numbers, for example

```
  ⎕ ← '⍞'⌹'Name' '          ' 'Street' ' ' 'Town' ' '
```

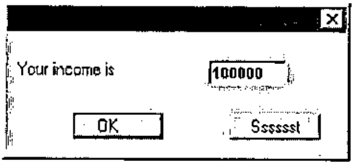
results into (notice that for sufficient size I gave street a suggestion of 10 blanks):



And of course, if you are not satisfied with the standard [OK] and [Cancel] buttons, you can specify your own in the left argument. For example:

```
  ⎕ ← '#' 'OK' 'Ssssst'⌹'Your income is' 100000
```

to get:



Your income is 100000

Detailed layout

My experience is that the capabilities of the above gui-quad is quite satisfactory and is sufficient for most of my needs. Nevertheless there are occasions when I want more control over the layout. Now I can resort to ⎕*WI* but I still have to specify more nitty-gritty than I care for.

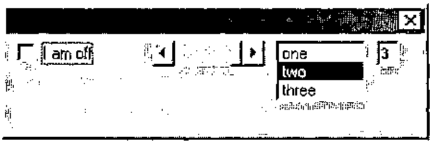
When I learned Fortran and PL/I (long, long ago) layout specifications for output were much simpler. For example to set three numbers on the printer in a field of size 10, I specified 3F10; for a text of size 12 I specified A12, for two empty lines I specified SKIP(3) (meaning 1 for next line and 2 more for 2 empty lines) and to position something, text or numbers, at position 35 I specified COL(35). Now is this so very different from how I want to "print" my dialogue box? Of course the needs were simpler at that time, but is the complexity of ⎕*WI* really needed? Wouldn't the simple Fortran or PL/I-like formatting be able to handle the majority of the graphical layout requirements sufficiently? I found that this type of layout specification could be used for gui layout rather well. Let me show you how to do this.

Again, the gui-quad gets its information on the right and returns it modified. We concentrate on the left argument that will have the Fortran-like specifications to describe the layout of each of the items in the right argument. As in Fortran, the specification will be in one vector (ehhhh, statement) separated from each other by a comma. But of course we will use compact symbolic and suggestive notation.

For example:

```
      ⎕ ← ',×I am off,-10,≡.one.two.three,#'∪ 0 1 2 3
 0 1 0.4 2 3
```

This displays the values 0, 1, 2 and 3 in layout-elements…
×: a clickbox with value off (describes layout for value 0) and text "I am off"
-: a slider of size 10 (describes layout for value 1)
≡: a list with alternatives "one"/"two"/"three" where "two" is on (describes layout for value 2)
#: a number field (describes layout for value 3)



As you see, this dialogue does not look very good, because without much instruction, gui-quad places all the elements next to each other. Now in Fortran and PL/I we also had positional instructions. In contrast to the other layout specifications that were associated with a value, positional instructions were self-reliant and not related to an input value. For example I could skip 4 positions using X4, and go to column 35 using COL(35). Apparently gui-quad also needs positional instructions. Of course, in APL we do not want the specifications as verbose as in Fortran or PL/I. We will use single-character symbols instead. The following 6 positional instructions take care of this:

`>#`: # positions to the right (here # stands for any number for example "`>3`")

`<#`: # positions to the left

`^#`: # lines up (this would be PL/I magic on the printer)

`v#`: # lines down

`/`: to the beginning of next line

`◊#.#`: to position #.#.

Positions are expressed in characters. For example "`◊5.9`" will place the next item at line 5, column 9. The size of a character is an average; for proportional fonts, the character "i" takes less space than character "m".

Now if we apply our new specifications we might specify

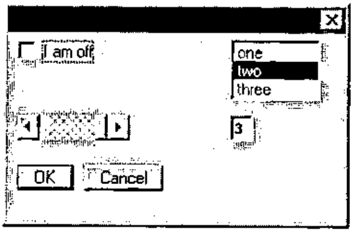
```
      ⎕ ← ',×I am off,v3,-10,◊1.20,≡.one.two.three,v,#'∪ 0 1 2 3
```

and the result looks much better.

One important feature is still missing: exit buttons. As with the positional instructions, exit buttons are not associated with information items from the right argument. They can be inserted in the left argument wherever it is convenient. They are characterized by the arrow (the suggestion is that they "jump"away). We use → for normal exit buttons, ↑ for the default button (pressing [Enter]), and ↓ for the [Esc] button. So...

```
   ⎕←',×I am off,∨3,-10,⋄1.20,⁎.one.two.three,∨,#,/2,↑ OK ,↓Cancel'⌹0 1 2 3
```

will generate the following dialogue box:



☐ I am off

one
two
three

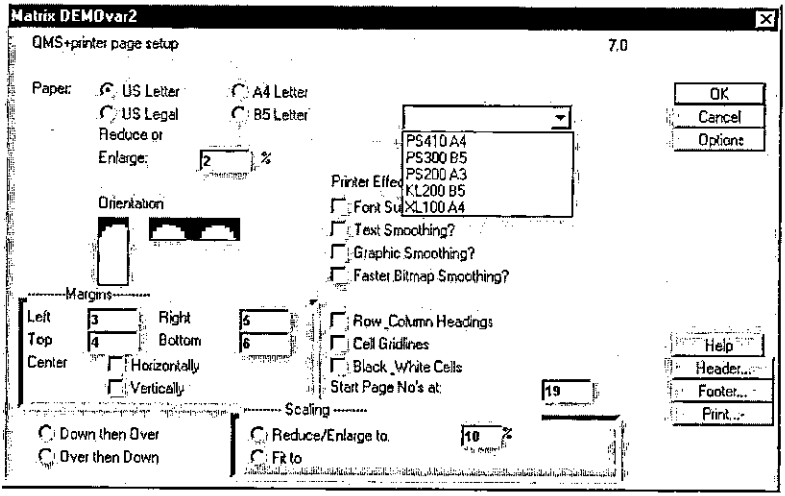
3

OK Cancel

and may return:

```
   2 0 1 2 3
```

When you want a very complicated box, the left argument of gui-quad is relatively more complicated, of course. For example to get the box…



Matrix DEMOvar2

QMS+printer page setup 7.0

Paper: ● US Letter ○ A4 Letter ○ US Legal ○ B5 Letter

Reduce or Enlarge: 2 %

Orientation

PS410 A4
PS300 B5
PS200 A3
KL200 B5
XL100 A4

Printer Effe…

☐ Font Su…

☐ Text Smoothing?

☐ Graphic Smoothing?

☐ Faster Bitmap Smoothing?

Margins

Left 3 Right 5

Top 4 Bottom 6

Center ☐ Horizontally ☐ Vertically

☐ Row_Column Headings

☐ Cell Gridlines

☐ Black_White Cells

Start Page No's at: 19

○ Down then Over

○ Over then Down

Scaling

○ Reduce/Enlarge to. 10 %

○ Fit to

I had to make a matrix *DEMOvar2* with the following text:

```
Matrix DEMOvar2, ⍝ This will be the title
⋄1.2,¨ QMS+printer page setup, ⍝ Comma between item & comment
⋄1.59,¨ 7.0,
⋄3.2,¨ Paper:,
⋄3.9,○US Letter,         ⍝ 4 radio buttons
∨,○US Legal,
⋄3.22,○A4 Letter,
∨,○B5 Letter,
⋄5.9,¨Reduce or,
∨,¨Enlarge:,
⋄6.19,⎕5,                ⍝ editable field of size 5
⋄6.25,¨%  ,
⋄8.9,¨Orientation,
∨⌸3.3.DmGUIOpicSNOOP,    ⍝ just for demonstration (too
>1,⍠1.9.DmGUIOpicSNOOP,  ⍝ small)sizes of picture
⋄13.2,¨Left,
⋄13.8,⎕5,
⋄14.2,¨Top,
⋄14.8,⎕5,
⋄13.15,¨Right,
⋄13.23,⎕5,
⋄14.15,¨Bottom,
⋄14.23,⎕5,
⋄15.2,¨Center,
⋄15.10,×Horizontally ,   ⍝ Two click-boxes
∨,×Vertically  ,
⋄18.3,○Down then Over,
∨,○Over then Down,
⋄18.45,⎕5,
⋄18.49,¨%,
⋄18.24,○Reduce/Enlarge to.,
∨,○Fit to,
⋄7.32,¨Printer Effects.,
∨,×Font Substitution?,
∨,×Text Smoothing?,
∨,×Graphic Smoothing?,
∨,×Faster Bitmap Smoothing?,
∨,×Row & Column Headings,
∨,×Cell Gridlines,
∨,×Black & White Cells,
∨,¨Start Page No's at:,
⋄16.53,⎕5,
⋄ 3.66,↑   OK    , ⋄4.66,↓ Cancel ,⍝ Exit buttons
⋄5.66,→ Options, ⋄14.66,→  Help  , ⋄15.66,→Header...,
⋄16.66,→Footer...,⋄17.66,→Print...-,
⋄4.39,≡ChoosePrinter.PS410A4.PS300B5.PS200A3.KL200B5.XL100A4,
⋄12.1,⎕5.30.----------Margins---------, ⍝ Frames with text
⋄17.22,⎕3.39.---------- Scaling ---------,
```

and used it:

```
    ⎕←DEMOvar2⌼ 1 2 3 4 5 6 0 0 9 10 11 0 0 0 0 0 0 0 19 0
```

You see in this example that gui-quad accepts a matrix as well as a vector as left argument. You can also see that I can give a comment in the specifications. Now you might think that this matrix is fairly complex. However, keep in mind that it defines a dialogue box with 57 items and that only 57 short lines are sufficient to define this dialogue box.

Progress bars

Frankly speaking, at this point I have explained everything that should be known about the basics of my proposed gui-quad. I could describe a few more features that are available (such as the ¨-specification that you can sprinkle around in the left argument to put explanatory texts in the dialogue box, or the ⎕-specification for frames). Or I could clarify more details of the functions mentioned (such as the number right after the ≡-specification that instructs how many lines are available). This I will not do. For those that are interested I refer instead to the example function and the description that I wrote.

Unfortunately I could not resist extending the functionality of gui-quad in a new way. This was based on my observation that for progress bars the simple dialogs were no problem, but as the need for more sophisticated progress-bar-dialogs grew, the required functions for those dialogs were always purely gui-quad functions. For that reason I finally decided to put the progress bar functionality into the gui-quad .

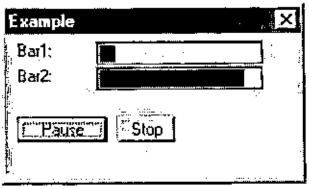
Now, using the gui-quad for progress bars differs in one aspect essentially from its normal use. Normally the quad-gui is called only once; it waits for the user to make its changes and only after the user clicks an exit button it gives back control.

For progress-use however, the gui-quad relinguishes control immediately; therefore it requires three different types of invocation:

Initially a one-time call that brings up the window. This first call fully describes the window layout in the normal gui-quad way using the initial values of the right argument in a way that is specified in the layout description of the left argument. For example:

```
      ⎕←'Example,¨Bar1:,"15,/,¨Bar2:,"15,//,↑Pause,→Stop'⌼ 0 1
 ¯1 0 1
 ⍝ ¯1  means no buttons clicked
 ⍝ 0 1 are values of progress bars
```

Next, the gui-quad is called repeatedly to show the new, updated values in its window. Now there is no layout at the left because the layout was already specified in the initial call. At each such call the new progressbar values will update the progressbar and the program checks the other gui-items (if present) to see if the user changed them. In our example the user could have pressed the Pause-button or the Stop-button or the [X] in the top-right corner.



For example

```
     ⎕"'"'⌼ 0.1 0.9     ⍝ call one
¯1 0.1 0.9
     ⎕"'"'⌼ 0.2 0.8     ⍝ call two
2 0.2 0.8               ⍝ 2 means "second exit button"
     ⎕"'"'⌼ 0.3 0.7     ⍝ call three
0 0.3 0.7               ⍝ 0 means [X] was clicked on
```

In this example you see that the result in the second call reports that exit button 2 was pressed (2 instead of ¯1) and the last call reports that the upper-right close button [X] was pressed.

Finally a call that removes the dialogue. Its syntax is:

```
     '⍋'⌼ ⍳0
```

In this example we only applied the exit buttons, but as you might expect, you can use all the previously mentioned gui-items that are available in gui-quad.

Discussion

The biggest forte of this gui-quad is that its arguments concentrate on the essentials. There is no programming "fuzz" that you need with ⎕*WI*. No forms, no opening, closing, showing, waiting, window sizing, deleting, on-methods, and no properties stuff.

As a consequence the gui-quad arguments are very compact. A vector of one or two lines easily specifies a dialogue that needs one or more pages of code using ⎕*WI*. Therefore it is quickly programmed, read fast by the programmer and quickly debugged or reprogrammed. Probably you might not see this as a beginner because the funny symbols in the left-argument specifications could look a bit daunting at first, but using gui-quad for a while will convince you of its advantage.

Of course the layout specification of variable *DEMOvar2* was a lot more complicated than the other examples. This is not surprising as the generated dialogue is more complicated too. The essential question is of course: are the specifications for complicated windows easier with ⎕*WI*?

My answer is: no. The reverse is true: both for simple and for complicated windows the gui-quad is much easier than the ⎕*WI*. You only need to provide a minimum of layout specifications. If your dialogue is very complicated, the specifications are more elaborate, but the amount of overhead is very small.

Of course gui-quad can not replace ⎕*WI* altogether. There are luxury features that are available in ⎕*WI* and not in gui-quad. So for an occasional really sophisticated window you have to "fall back" to the ⎕*WI*. Or use CAUSEWAY [2] which is also much more powerful than this gui-quad and still takes care of a lot of nitty-gritty that is needed with ⎕*WI*. However my experience is that for most applications I am quite happy with the available functions in the gui-quad.

Conclusion

Several times I have spoken about my experiences with gui-quad. So you might wonder where gui-quad is implemented. Unfortunately nowhere, yet. My experience is based on a user function that we wrote and which acts like the proposed gui-quad.

As I hope to have shown, the gui-quad is a very powerful, useful and relatively easy way to interact with the user. So I plead for adding it to the APL primitives.

As long as the gui-quad is not available as a primitive we have to resort to a user function. Such a function, called mGUIO (Graphical User Input Output), is currently written for APLX/Win, APLX/Mac, APL2000 and APL2C. If somebody is interested he or she could contact the first author.

Literature

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

A.Smith (1995): Causeway workshop: Pitkospuut GUIsuon Yli. Vector 11(5)60-64. See also: http://www.causeway.co.uk/

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	Crossley, Peelle, Thomson	90
Perfect Printing	Stephen Taylor	95
A Girl's Best Friend	Phil Chastney	109
Dyalog.Net: APLScript and Things Textual	John Daintree	114

TECHNICAL CORRESPONDENCE

Vol 19.1: Bitmaps in Dyalog APL/W

From: David Crossley 28th July 2002

I would like to make three points regarding my article in Vector 19.1.

Firstly, I wish to correct the record in that Dyalog APL/W can now interface fully with Bitmaps represented in 24-bit colour using the CBits property whose data element is an integer matrix of the same size as the Bitmat object giving the encoded RGB components for each pixel. This was implemented later than Version 8.1. 3 on which I based my article. However, the discussion regarding the specification of Bitmap structures, which are relevant also to the data structures for Cursors and Icons, may still be of general interest.

Secondly, I wish to correct an error in a code fragment near the beginning of the article (entirely the fault of the author) which read as follows:

```
bits←0≠'ff.bm'⎕WG'Bits' 'CMap'
```

You will of course realise that the CMap parameter should not be there.

Thirdly, a whole Appendix was omitted from the article (not the fault of the author) which contains the all-important specification of the Bitmap file structure. This may well appear on the Internet site for Vector 19.1, though not available when I wrote this letter, but for readers preferring paper copy it is reproduced below.

Appendix

```
      ∇ r←BM_Read f;Bits;CMap;h;hdr;i;j;n;t;w;z

 [1]    ⍝∇ Read and define Bits and CMap from Bitmap file f

 [2]    ⍝  r ←→ 1(<LM/IM/NM Bits><NM CMap>) if successful,
                    or 0 ((0 0⍴0)(0 3⍴0)) if failed
 [3]     →Start
```

```
[4]   ⍝ Bitmap file structure...
[5]
[6]   ⍝ BITMAPFILEHEADER
[7]   ⍝ 1  0-1   UINT  BM indicates Bitmap
[8]   ⍝ 2  2-5   DWORD Total file size in bytes
[9]   ⍝ 3  6-7   UINT  0 - reserved
[10]  ⍝ 4  8-9   UINT  0 - reserved
[11]  ⍝ 5  10-13 DWORD Offset to bits from beginning of file
[12]
[13]  ⍝ BITMAPINFOHEADER
[14]  ⍝ 6  14-17 DWORD Size of this structure in bytes
[15]  ⍝ 7  18-21 LONG  Width of bitmap in pixels
[16]  ⍝ 8  22-25 LONG  Height of bitmap in pixels
[17]  ⍝ 9  26-27 WORD  Planes (set to 1)
[18]  ⍝ 10 28-29 WORD  Colour bits per pixel (1,4,8 or 24)
[19]  ⍝ 11 30-33 DWORD Compression scheme (or 0)
[20]  ⍝ 12 34-37 DWORD Size of bitmap in bytes (0 unless compr scheme used)
[21]  ⍝ 13 38-41 LONG  Horizontal resolution in pixels/metre
[22]  ⍝ 14 42-45 LONG  Vertical resolution in pixels/metre
[23]  ⍝ 15 46-49 DWORD Nr of colours used (supersedes Colour bits info)
[24]  ⍝ 16 50-53 DWORD Nr of important colours in image
[25]
[26]  ⍝ RGBQUAD 4 bytes per colour (abseht if Colour bits is 24)
[27]  ⍝           BYTE  Blue intensity  |
[28]  ⍝           BYTE  Green intensity | Note order, ie NOT RGB
[29]  ⍝           BYTE  Red intensity   |
[30]  ⍝           BYTE  Set to 0
[31]
[32]  ⍝ Bitmap Bits, rows padded to 4-byte boundary, from BOTTOM row to top..
[33]  ⍝ 1-bit     1 bit/pixel (row width÷8 bytes rounded to 4-byte boundary)
[34]  ⍝ 4-bit     4 bits/pixel (row width÷2 bytes etc)
[35]  ⍝ 8-bit     1 byte/pixel
[36]  ⍝ 24-bit    3 bytes/pixel in order B-G-R (no colour map)
[37]
[38]  Start:
[39]   r←0((0 0⍴0)(0 3⍴0))
[40]   :Trap 0
[41]       :If 'BM'≢2↑z←biread f ⋄ :Return ⋄ :EndIf
[42]       hdr←16⍴0
[43]       hdr[2 5 6 7 8 11 12 13 14 15 16]←2 BM_WORD
              z[(2 10 14 18 22 30 34 38 42 46 50)∘.+⍳4]
[44]       hdr[9 10]←BM_WORD z[26 28∘.+⍳2]
[45]       i w h n←hdr[5 7 8 15]
[46]       :Select 10⊃hdr
[47]       :Case 1
[48]           n+←2×n=0
[49]           CMap←¯1+⌽⎕AV⍳0 ¯1↓n 4⍴z[54+⍳4×n]
[50]           t←⎕NXLATE 0 ⋄ ((⍳256)-⎕IO)⎕NXLATE 0
                 ⍝ to avoid translation in byte→bit conversion
[51]           Bits←⊖h w↑(h,32×⌈w÷32)⍴11 ⎕DR i↓z
[52]           t ⎕NXLATE 0
[53]       :Case 4
[54]           n+←16×n=0
[55]           CMap←¯1+⌽⎕AV⍳0 ¯1↓n 4⍴z[54+⍳4×n]
[56]           Bits←⊖h w↑(h,8×⌈w÷8)⍴⍉16 16⊤¯1+⎕AV⍳i↓z
```

```
          :Case 8
              n+←256×n=0
              CMap←¯1+⌽⎕AV⍳0 ¯1↓n 4⍴z[54+⍳4×n]
              Bits←¯1+⎕AV⍳h w↑(h,4×⌈w÷4)⍴i↓z
          :Case 24 ⍝ convert to use colour table if possible
              Bits←⊖h w⍴(3⍴256)⊥¯1+⎕AV⍳⊖⍉((h×w),3)⍴(h,w×3)↑(h,4×⌈w×0.75)⍴i↓z
              :If 256≥⍴CMap←∪,Bits
                  Bits←¯1+CMap⍳Bits
                  CMap←⍉(3⍴256)⊤CMap
              :Else
                  CMap←0 3⍴0
              :EndIf
          :EndSelect
          r←1(Bits CMap)
      :EndTrap
    ∇

    ∇ r←{a}BM_WORD b
     ⍝∇ Convert WORD/DWORD from text to integer, or vice-versa
     ⍝  a ←→ IS     {←1} 1:WORD, 2:DWORD
     ⍝  b ←→ TV/TM Each 2/4 bytes represents a WORD/DWORD, or
     ⍝         IS/IV integer to text representedation of a WORD/DWORD
     ⍝  r ←→ IV or TV
     ⍝  Note: WORD and DWORD are big-endian
      :If 0=⎕NC'a' ⋄ a←1 ⋄ :EndIf
      a←2×1+a≠1
      :If 82=⎕DR b ⋄ r←(a⍴256)⊥¯1+⎕AV⍳⍉⌽((⌊(⍴,b)÷a),a)⍴b
      :Else ⋄ r←⎕AV[⎕IO+,⌽⍉(a⍴256)⊤,b]
      :EndIf
    ∇

    ∇ r←biread n;t
     ⍝∇ Binary read: Read file without translation
      n ⎕NTIE t←¯1+⌊/0,⎕NNUMS
      ((⍳256)-⎕IO)⎕NXLATE t
      r←⎕NREAD t,82,(⎕NSIZE t),0 ⋄ ⎕NUNTIE t
    ∇
```

Second-order Josephus

From: Howard A. Peelle (hapeelle@educ.umass.edu) July 2002

It was nice to see Eugene McDonnell's "At Play with J" column in Vector 18:4 on "Second-order Josephus" which relates to previous articles in Vector on the Josephus problem. Hopefully there will be more in the future.

Gene cited Boyko Bantchev who recalled the neat verb to compute the Josephus survivor number:

```
  S =: 1&|.&.#:
```

The reader should be cautioned that the input and result are in origin 1 while J uses origin 0, and that this verb does not generalize to the original Josephus problem (using a killing interval of three) nor to other intervals.

Bantchev proffered J verbs to solve the Second-order Josephus problem, beginning with a verb to eliminate item S #y from any list y which, if restricted to unique items (as needed here), can be written more succinctly as:

```
  E =: -. <:@S@# { ]
```

This can be embodied in a master verb using a gerund to find the Second-order survivor number by simulation:

```
  S2a =: E ^: (<: ` Natural)
          Natural =: >:@i.
```

Interested in producing a sequence of such results directly, Gene offered a "high-speed" verb to compute all but one last item using a power of 2 needed for input. I've rewritten it tacitly here to produce the full sequence:

```
  J2t =: Sort @ (Natural@Power , Copy@Power@i.)
          Sort =: /:~
          Natural =: }.@i.@>:
          Power =: 2&^
          Copy =: # +:
```

The following alternative definition is more efficient without sorting:

```
      J2r =: >:@Copies # Natural@Power
          Copies =: Power@i. In} Power # 0:
               In =: <:@+:@[
```

An even more efficient definition is:

```
J2b =: +/\ @ Bits =: 1: , ;@:(Take each)@Power@i.
    Take =: +: {. (# 1:)
```

This verb runs about 8 times faster than Gene's hsJ2 in about 30% space for a sample input of 12 (using J4.06a on a Pentium II PC running Windows 95 with 256M).

Gene's column could have concluded with a verb to solve the Second-order Josephus problem:

```
S2c =: <: { J2x @ Enuf =: >.@Log =: 2&^.
```

But, of course, this is redundant. Instead, Bantchev offered a verb S2 using base-2 encoding to solve the problem (except for an input of 1). Here is an alternative inclusive definition:

```
S2p =: (] <. 1: + (- -:)) <.&.(2&^.)
```

This definition produces the Second-order Josephus sequence about 6 times faster than Bantchev's S2 for 2+i.30 and generally runs even faster (but in increasingly more space) for larger inputs. (But who would want to do so much killing?!)

From : Norman Thomson 5 September 2002

Regarding Phil Chastney's letter in Vector 19.4, in comparing 16 words with 12 pages, I am of course employing a degree of exaggeration for effect – but not much! Occam's razor is about minimizing concepts, which is precisely what rank does in providing a unifying *principle* which is absent, or at best only qualifiedly present, in APL, a point amply demonstrated by Phil himself in making the case for redefinition of some of the APL primitives. I recall the disappointment I felt when first coming to APL when it dawned on me that default axes were a matter for the whim of the software producer. Rank does not in itself remove problem complexity, but with J, at least I know that I must ascribe my struggles with domain-extended problems solely to personal weakness of intellect (as demonstrated by Howard's improvements on some of my illustrations!), uncompounded with arbitrary rules set by the product provider.

Perfect Printing

by Stephen Taylor
Email: sjt@lambenttechnology.com

Abstract

APL.RTF creates, previews and prints Windows documents of arbitrary length to the standard of an advanced Microsoft Word user. The resulting files can be opened, edited or printed from Microsoft Word, and can also be attached to email messages. The APL application-level code required to use this is terse, with large-scale mapping from APL nested lists and tables to document paragraphs and tables. Execution is fast, and the APL footprint is small.

A workspace containing much of what is described in this article has been posted on the *Vector* web site. You might find this article interesting for the way in which object-oriented and functional programming layers have been combined, inspired by something seen in a J programming lab. Or you might want to use APL.RTF, in which case contact the author.

Background

Once our clients were content for letters and other documents to look like a computer had printed them. Those days are gone. Now they want their documents to look as good as if produced by a competent secretary with a word processor.

Printing through standard RTF tools gets some control over fonts, alignment and so on. But it doesn't meet the 'competent secretary' standard. We needed to produce letters and forms with tables, frames, boxes, shaded areas, hairlines, arbitrarily positioned text, switches between portrait and landscape orientations and so on.

From one point of view, a solution is available without any programming at all. We can call Microsoft Word from within APL and do anything Microsoft Word can do. This works but has two serious drawbacks. The first is that it runs slowly. The second is that, while one gets all of Word's awesome abilities, including searching and replacing, the APL application code has too much to do in managing Word.

If we could keep most of Word's *presentation* controls, we would sacrifice most of its *editing* abilities and just append content sequentially to a document. In particular, we should like our application code to map straight from large-scale APL data structures, such as nested tables and lists of text strings, to tables and paragraphs in a document, without stopping to specify every detail of presentation and to hide any encoding or markup done to achieve this.

We did it. APL.RTF is a printing subsystem with a modest footprint (135Kb in Dyalog APL 9 under Windows 98SE) that fits cleanly into an application. It enables the application to produce RTF (Rich Text Format) documents of arbitrary length that can be printed or edited by Word, or sent as email attachments. It runs fast, exploiting the Microsoft type libraries to do the hard work. We believe it could be ported to J or K fairly easily. It supports more presentation features than most Word users know how to employ, and enables our applications to meet or surpass the 'competent secretary' criterion.

Word-processing presentation features supported

The interface supports control of the following features:

Character properties

- font family
- bold and italic styles
- type size
- type colour
- *kerning* (inter-character spacing)
- special characters: checkboxes, current date, left & right quotes, line break, page break, page number, tab

Paragraph properties

- left, right, centered or justified alignment
- paragraph and first-line indentation
- *leading* (spacing before and after paragraphs)
- *keep* to discourage Word from breaking a paragraph across a page boundary

- *keep-with-next* to discourage Word from separating two paragraphs with a page boundary
- page breaks
- bulleted lists
- tab stops: left, right, centred and decimal; with various leads

Tables

- column heading rows that repeat when the table crosses a page boundary
- row-level control of *keep* and *keep-with-next* properties
- column- or cell-level control of cell width
- cell-level control of alignment
- cell-level control of *leading* before and after
- cell-level control of shading
- cell-level control of top, left, bottom & right border widths and colours
- horizontal merging of adjacent cells

Pictures

- bitmap picture files

Inline styling

- bold and italic
- expanded inter-character spacing
- tab stops
- arbitrary positioning of frames
- frame size
- frame position and alignment, choice of reference frame
- text wrapping and separation, overlays
- overlapping frames
- border width, colour and padding

- background shading

Section properties

- US letter or A4 paper size
- portrait or landscape orientation
- multiple columns
- margin sizes
- page header and footer text, with distinct headers and footers for the first page of each section
- header and footer offsets

A quick tour

Creating, writing to and printing a document

Writing a document from APL.RTF is a lot like writing a Microsoft Word document. You start by creating a new blank document from a template. You add paragraphs, assigning them styles from the template's style sheet. The style sheet specifies different named paragraph styles in terms of such things as alignment, indents, fonts, colours and leading. You also add lists, again giving them styles from the style sheet.

You add tables, with exact control of their appearance: alignment, column widths, hairlines, text styles, shading, merged cells.

Just as in Microsoft Word, you control page layout: paper size, orientation, margins and the offsets of the headers and footers from the page edge. Your document can have multiple sections with different page layouts, including switching between orientations and multiple columns.

You can also use absolute positioning to place text or pictures precisely where you want them on the page, floating it above the other text or having the main text wrap around it. Your text boxes can have background shading or borders.

When you've finished your document, tell it to print itself, or show you a print preview. Or tell it to close itself and return the name of an RTF file you can print or attach to an email.

This is particularly valuable when developing an application. Start with a template document similar to what you want, write output to it, and immediately preview it onscreen. Then you can fine-tune appearance and positioning by editing either the application script or the template's style sheet, or both.

The Document class

The core of APL.RTF is its Document class, represented by the *Document* namespace. It contains further namespaces, which correspond to Microsoft Word template documents for different purposes. As with Microsoft Word, the default template is called Normal.

The *Document* class contains its own methods (functions), and a constructor method (*New*) for creating instances of itself. The *New* method returns a copy of the template into which has been copied all the *Document* methods not already defined in the template. In this way, the templates inherit methods from the *Document* class.

This is not so for properties: new instances do not inherit property values (variables) from *Document.* Instead they are created by the *Initialise* function. When *New* has created an instance, it invokes *Initialise*, which was inherited either from the template or from the *Document* class.

Initialise sets the new document's properties. Some of these are private to the document and are not intended for reference by the programmer. Others control key aspects of presentation. Chief of these are the page layout, font table and style sheet.

The *font table* is a list of font family/name pairs, eg 'swiss Arial' or 'roman Times New Roman'.

The *style sheet* is a table of RTF style definitions used to assign styles to paragraphs. The *Document* method *makeStyleSheet* creates a default style sheet, for the *Normal* template. It adapts from the *RTF.STYLEMACROS* namespace a specialist vocabulary for defining style sheets. You can adapt *Document*'s *makeStyleSheet* function for new document templates for your applications.

The *page layout* is a list of name-value pairs that corresponds to the key features controlled by Microsoft Word's File/Page Setup dialogue, eg paper size and orientation, and margin sizes.

Paragraphs, lists and tables are added to the document using its methods *AddPara, AddList* and *AddTable*. The method *AddBox* sets paragraphs and lists on the page using 'absolute positioning'. Absolute positioning is very powerful and allows you to place a frame of text or graphics anywhere on the page, with optional borders or background shading, and to determine whether the background text will flow around it or lie underneath it.

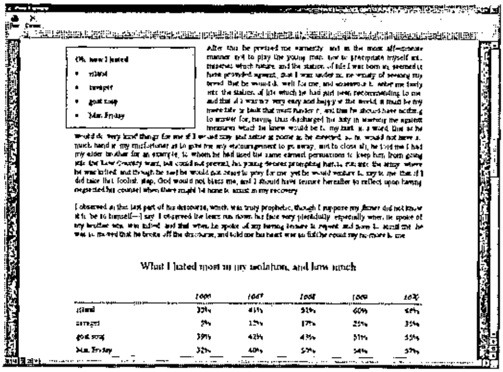
The style of paragraphs, lists and tables is controlled by references to the style sheet. These references apply defined styles to entire paragraphs, list items or table cells. APL.RTF also supports 'inline styling', in which part of a string of text can be assigned bold, italic or expanded-text style, or some combination of them. This is provided by document methods *EMBOLDEN*, *ITALICISE* and *EXPAND*.

The document method *Print* sends the document to Microsoft Word for printing.

The document method *Close* acts like *Print*, but returns the name of the RTF file instead of sending it for printing. This file is then available for use as, say, an email attachment.

Print Previews

The *UTILS* namespace contains a function *preview*, which will display an RTF file in a browser, exactly as it would look in Microsoft Word. (It uses Microsoft Web Browser, the same OCX control that Internet Explorer uses.) If the user confirms, the document is printed; else destroyed.



Oh, how I hated

- island
- savages
- goat soup
- Man Friday

After this he pressed me earnestly, and in the most affectionate manner, not to play the young man, nor to precipitate myself into miseries which nature, and the station of life I was born in, seemed to have provided against; that I was under no necessity of seeking my bread; that he would do well for me, and endeavour to enter me fairly into the station of life which he had just been recommending to me; and that if I was not very easy and happy in the world, it must be my mere fate or fault that must hinder it; and that he should have nothing to answer for, having thus discharged his duty in warning me against measures which he knew would be to my hurt; in a word, that as he would do very kind things for me if I would stay and settle at home as he directed, so he would not have so much hand in my misfortunes as to give me any encouragement to go away; and to close all, he told me I had my elder brother for an example, to whom he had used the same earnest persuasions to keep him from going into the Low Country wars, but could not prevail, his young desires prompting him to run into the army, where he was killed; and though he said he would not cease to pray for me, yet he would venture to say to me, that if I did take this foolish step, God would not bless me, and I should have leisure hereafter to reflect upon having neglected his counsel when there might be none to assist in my recovery.

I observed in this last part of his discourse, which was truly prophetic, though I suppose my *father* did not know it to be so himself—I say, I observed the tears run down his face very plentifully, especially when he spoke of my brother who was killed; and that when he spoke of my having leisure to repent, and none to assist me, he was so moved that he broke off the discourse, and told me his heart was so full he could say no more to me.

What I hated most in my isolation, and how much

	1666	*1667*	*1668*	*1669*	*1670*
island	33%	41%	51%	60%	68%
savages	3%	12%	17%	25%	35%
goat soup	39%	42%	43%	51%	55%
Man Friday	32%	40%	57%	54%	57%

In fact, the *Print* method also tests whether it is in runtime mode. If not, it passes *PrintPreview* a snapshot copy of the document's current state. Developers always get a preview, but the active document is untouched. This allows developers to add to a document and immediately see the effect without destroying the document.

The SCRIPTS namespace

The *SCRIPTS* namespace contains the application-level code. In this context, a script is an APL function that adds content to a document, specifying its presentation attributes as it does so.

Four scripts are called by the *Examples* function in the workspace root: these demonstrate different capabilities of APL.RTF. Three scripts are called by the *SVQ* function in the workspace root: these demonstrate some documents created for a commercial application.

```
 ∇   Examples scripts;doc;script
[1] ⍝ scripts is a list of numbers in range 1-4; see below
[2] ⍝ Printer is name of a Windows printer
[3]
[4] doc←#.Document.(New Example)
[5]
[6] doc.Started←0 ⍝ 1st script not to start new section
[7]
[8] :For script :In scripts
[9]:Select script
[10]    :Case 1 ⋄ #.SCRIPTS.GENERAL.Script doc    ⍝ general
[11]    :Case 2 ⋄ #.SCRIPTS.MULTI.Script doc      ⍝ multi cols
[12]    :Case 3 ⋄ #.SCRIPTS.ORIENT.Script doc     ⍝ 2 orientns
[13]    :Case 4 ⋄ #.SCRIPTS.POSN.Script doc       ⍝ abs posn
[14]    :EndSelect
[15] :EndFor
[16]
[17] doc.Print Printer 1
 ∇
```

You can see from `Examples` and `SVQ` that the scripts are passed a reference to a document object, to which they add their output. They can be called in any order: compare

```
Examples 1 2 3 4
Examples 4 3 2 1
```

The RTF namespace

> "[Object Oriented Programming] is effective. Unfortunately, some people think that if some OOP is good, then more is better. It is a mistake to think of OOP as an alternative to FP (functional programming); all languages, at the low level, have FP, and at a higher level, have OOP."
>
> *Object Oriented Programming lab in J*

The RTF namespace contains the functional programming layer used by the Document class. It is mostly concerned with the encoding of RTF control words. (Enthusiasts might appreciate the very fast D functions used by `TABULATE` to convert nested tables into RTF code strings.) Application programmers should never need to refer to it.

The Microsoft type libraries

APL.RTF delegates much processing to the Microsoft type libraries. The application programmer need do nothing to obtain or invoke these but ensure that users have Microsoft Word installed on their PCs.

Five type libraries are used: Microsoft HTML Object Library, Microsoft Internet Controls, Microsoft Word Object Library, Microsoft Office Object Library and Visual Basic for Applications Extensibility.

The APL.RTF workspace is distributed with these libraries unloaded. Dyalog APL will load them on first use; they occupy nearly 1Mb of the workspace. Since loading the libraries causes a noticeable delay, applications are best saved with the libraries already loaded.

Creating a new document template

APL.RTF Document templates correspond exactly to Microsoft Word template documents. That is, a template is used to create various documents that share a common look.

This common look is embodied in the document's style sheet, which names, numbers and describes a list of paragraph styles. Paragraph styles are described in such terms as the font family and size used, bold or italic style, leading before or after the paragraph, alignment and indentation.

Many users of Microsoft Word find it convenient to set the presentation attributes of their document's paragraphs on an ad-hoc basis. Not so the APL.RTF application programmer, who wishes to keep this detail out of the application's script. Rather than specify presentation attributes 'in line' in his script, he lists them in the document template's style sheet.

There are a handful of exceptions to this. The document's *EMBOLDEN*, *ITALICISE* and *EXPAND* methods can be used to apply this 'in line' styling to words and phrases. Also, the position and leads for tab stops can be set in the script. See the *SVQ* scripts for examples where tab stops with underlined leads are used to create places to write on a form.

As in Microsoft Word, the default document template is called Normal. It is represented by an empty namespace, *Normal*. Because it is empty, it inherits all its methods from its parent class/namespace, *Document*. One of these methods, *Initialise*, sets its properties.

Initialise calls two functions of interest to the application programmer: *makeStyles* and *makeLayout*. The former creates the document's style sheet, and font- and colour tables. The latter specifies the document's page layout, corresponding roughly to the attributes controlled in Microsoft Word by the File/Page Setup dialogue.

To create a new document template, create a new namespace within *Document* and copy into it either or both of *makeStyles* and *makeLayout*. Edit those functions to suit your application.

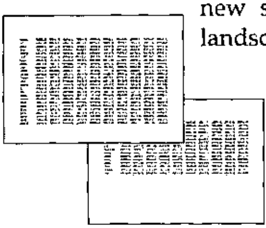
Your new template needs in it only what is to differ from *Normal*. It inherits everything else from its parent, *Document*.

Samples and examples

The *GENERAL* script shows various APL.RTF features. The *AddPara* appends paragraphs of text; wrapping and justification are managed by APL.RTF.

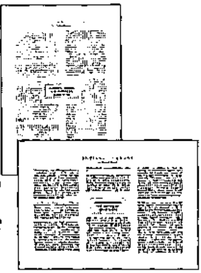
```
AddPara'Sched Title' 'Robinson Crusoe'
AddPara'Subtitle' 'by Daniel Defoe'
AddPara('Body Text')(#.DATA.texts.(p1 p2 p3))
```

A couple of pull quotes are hung in boxes against the margins, the body text wraps easily around them. A table and logo are shown. A new section opens, and the orientation switches to landscape.



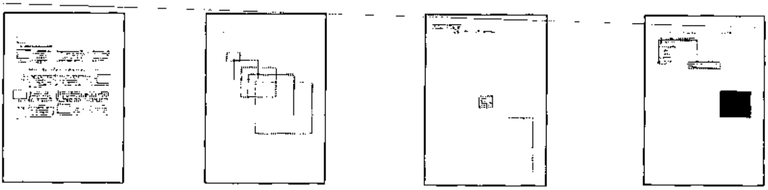
A large table is appended. Overflowing its first page, its header rows are repeated automatically on continuation pages.

The *MULTI* script shows the same story in a 2-column format, complete with a centred pull quote. The *ORIENT* script does the same again in the form of a 'scriptlet', then switches the page orientation to landscape and executes the scriptlet again with 3 columns specified. The scriptlet's output appears correctly in the landscape orientation. Even the page number, set by tab at ½" outside the right, adjusts to the changed page width. Here is the application code: script and scriptlet.

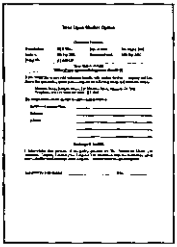
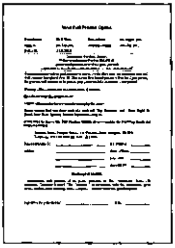
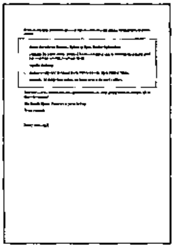
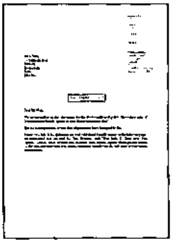


```
 ∇   Script doc
[1] ⍝ The scriptlet is independent of page orientation
[2]
[3]  doc.PageSetup'Orientation' 'Portrait'
[4]  doc Scriptlet 2 ⍝ cols
[5]
[6]  doc.NewSection'D'
[7]  doc.PageSetup'Orientation' 'Landscape'
[8]  doc Scriptlet 3 ⍝ cols
 ∇

 ∇   doc Scriptlet cols;⎕PATH;pq;rtf;styles;tabstops
[1]  ⎕PATH←'#.SCRIPTS.MULTI #.UTILS'
[2]
[3]  :With 'doc'
[4]    ⍝ set R-aligned tab stop 1/2" beyond R margin
[5]tabstops←⊂(0.5+PageWidth)('R')
[6]SetFooter('Footer')(TAB,'Page ',PAGENO)(tabstops)
[7]
[8]    ⍝ title and first paras
[9]AddPara'Sched Title' 'Robinson Crusoe'
[10]    AddPara'Subtitle' 'by Daniel Defoe'
[11]
[12]   ⍝ section break and new layout
[13]    NewSection'N'
[14]    PageSetup('Columns')(cols)
[15]
[16]    pq←'I would be satisfied with nothing but going to sea'
[17]    AddBox(PqPosnC)(ComposePara'Pull Quote'pq)
[18]    AddPara('Body Text')(#.DATA.texts.(p1 p2 p3 p4 p5 p6))
[19]
[20] :EndWith
 ∇
```

APL.RTF has a powerful ability to place text at arbitrary positions on the page, with or without borders ('boxes') or background shading. Body text can wrap around these or not. The fourth example script demonstrates this.



Finally the scripts called by *SVQ* demonstrate these features used in a commercial application to produce correspondence.

Key features

Document methods are monadic or niladic, like those of Microsoft's Common Object Models. Like them, some have *overloads* defined; that is, they accept arguments in varying forms. See the scripts for illustrations.

Paragraphs

The *AddPara* method adds one or more paragraphs to the document, specifying a style from the style sheet, and optionally specifying tab stops. *AddPara* discards empty paragraphs.

```
AddPara'Sched Title' 'Robinson Crusoe'
AddPara'Subtitle' 'by Daniel Defoe'
AddPara('Body Text')(#.DATA.texts.(p1 p2 p3))
```

Tab stops

Tab stops can be specified simply as scalars—a horizontal offset in inches from the paragraph start—or as enclosed pairs or triples. As pairs: the offset and the alignment of text to the tab stop: left, centred, right or decimal. As triples: the offset, alignment and lead to the tab stop: dot, middle dot, hyphen, underline, thick, equal. Tab stops with underline leads are used to mark places for handwriting in the *SVQ* documents on this page.

```
stops←0.25 2.5(PageWidth'L' 'Ul')
entab←{TAB,ω,TAB,TAB}
AddPara('Left Flush')(entab'Insurance Company Name')(stops)
AddPara('Left Flush')(entab'Reference')(stops)
AddPara('Left Flush')(entab'Address')(stops)
AddPara('Left Flush')(entab'')(stops)
AddPara('Left Flush')(entab'')(stops)
```

Special characters

The document has niladic methods for special characters like *TAB* and *PAGENO*.

It also has a dyadic function *LINE*, which joins its arguments with a new-line character. So

```
⊃LINE/a b c
```

joins text strings *a b c* into a single paragraph with embedded new-line characters.

Bitmaps

The *BITMAP* method takes two arguments: a scaling factor and a file name, which should have a .BMP extension. In principle it should be possible to calculate the scaling factor to produce an image of a desired size from a bitmap file. In practice, it is far simpler to take a guess, preview the result, and converge on a suitable factor.

Lists

Microsoft Word supports a wide variety of auto-numbered lists; APL.RTF none. Auto-numbering of lists is useful for editing documents with a word processor but is of little use for our purposes.

APL.RTF supports bulleted lists through the *AddList* method. Its syntax mirrors that of *AddPara*. Paragraphs composed with *AddList* are given bullet points.

Tables

APL applications easily compose information in the form of tables. APL.RTF easily maps APL tables to document tables.

The *AddTable* method takes a 2-element argument. The second is a nested table (rank-2 array) of character strings. These character strings may include RTF control words embedded by the in-line styling methods *EMBOLDEN*, *ITALICISE* and *EXPAND*, but otherwise need no presentation information.

Presentation instructions are embedded in the first argument element, the format array. This is a rank-3 array of 18 planes of the same shape as the data table. Each plane controls a different attribute of the table's presentation. See the function *FmtBigTbl* in the *SCRIPTS.GENERAL* namespace for an example.

Sections

For short documents a single page layout suffices. Longer documents might need a section in landscape orientation, for example, or to restart page numbering. The *NewSection* method takes one of the following arguments:

⍬ new section, retaining current layout

'D' new section, restoring the document's default layout

'N' new section, without a page break

Boxes

The ability to place a text box at an arbitrary position on the page is one of APL.RTF's most powerful features. The box may be set with or without a border, so the result may appear simply as text placed at an arbitrary position. See for example, the script in *SCRIPTS.Q226*, which uses *AddBox* to write an address exactly where it will show through an envelope window.

AddBox, like *AddTable*, takes a 2-element argument.

The first element is a list of 18 numbers. To keep track of the semantics of this, the namespace *RTF.BOX* contains a default set and a vocabulary in which to specify them. See particularly how the script in *SCRIPTS.POSN* takes a copy of this object, resets some values to produce a default set for the script, then uses a series of 'Box' functions to produce slight variations, specifying only the changes from these defaults.

```
bpd←BoxPosDef #.RTF.BOX              ⍝ box posn defaults

      ∇   bpd←BoxPosDef boxposnobj
     [1] ⍝ box position default values for this application
     [2] ⍝ object contains vocabulary for setting parameters
     [3]
     [4]  'bpd'⎕NS boxposnobj
     [5]
     [6]  :With 'bpd'
     [7]      Hposn←column aligned left
     [8]      Vposn←para aligned inline
     [9]      Size←0.75 0.75 ⍝ width & height in inches
     [10]Wrap Overlay Absnooverlp←do dont dont ⍝ wrapping
     [11]Separation←10 10 ⍝ h and v, in pts
     [12]Pen Colour←0.5 0 ⍝ 1/2pt black border
     [13]Pad Shade←5 0 ⍝ 5pt padding, no background shading
     [14] :EndWith
      ∇
```

```
d.AddBox(Box10 bpd)(empty) ⍝ adds one box in the document

      ∇   r←Box10 bpd;b
     [1]  'b'⎕NS bpd            ⍝ copy default BOXPOSN values
     [2]  b.(Wrap←dont)         ⍝ wrapping
     [3]  b.Separation←0 0      ⍝ h and v, in pts
     [4]  r←b.Parameters
      ∇
```

The second element is, for once, not 'raw' APL character strings or tables, but a marked-up RTF string. This must be so, because a box can contain paragraphs or lists, which themselves need to be composed against the document's style sheet. *AddBox* can place them on the page, but does not know how to compose them.

The *ComposePara* and *ComposeList* methods have the same syntax as *AddPara* and *AddList*, but return marked-up RTF strings instead of adding the paragraphs or list to the document. These strings can be concatenated and the result used as the second element of the *AddBox* argument.

Conclusion

With APL.RTF we are able to map APL data arrays to well-presented documents with minimal application code. We quickly adapt existing templates to new applications, and can mockup a handful of new document designs in a morning. We use well-proven Microsoft code to participate in the device independence that Windows applications commonly enjoy; the documents we produce live in the same world of PCs and email that our users do. They like this and so do we.

A Girl's Best Friend

by Phil Chastney
email: philip_chastney @ yahoo.com

0. introduction

No significant domain algebra this time, just a suggestion for expanding the current class of statement separators by the addition of two more items, allowing better control over exception handling.

1. the present situation

1.1 APL functions consist of a sequence of lines. In addition, some implementations allows braces to be used to group lines into blocks.

Each line may have 0, 1 or many separate executable statements, followed by an optional trailing comment.

If a line has more than one statement, the statements are separated by a diamond.

1.2 Actually, the description of diamond as a "statement separator" is a misnomer. Other languages may describe the comma as a separator, but in APL we know it as a primitive function which *joins*, not separates, two arrays.

Like a number of other such functions, the diamond is really a constructor, which joins up statements into lines. Or, to put it another way, it is a function which takes executable statements as arguments, and delivers another executable.

(Contrast this with the use of use of ⊣ and ⊢ ("lev" and "dex") which achieve a similar effect by operating in the data domain. This only works, however, if every subexpression delivers a result lying within the data domain.)

2. the present mechanism

2.1 The only issue with these function/line/statement things is how to handle side-effects, the most obvious of these being embedded assignments. We need more flexibility in this area if we want to improve exception handling, and the key concepts (borrowed from the database world) are those of "commit" and "rollback".

When an expression like

```
    N←N-1
```

is executed, the interpretation proceeds from right to left. The rightmost occurrence of N picks up the current value of N, and the leftmost occurrence of N receives the updated value. This means that in a statement like

```
    RECIP←÷N←N-1
```

the value of N will have been permanently updated if the expression fails thus:

```
      RECIP←÷N←N-1
DOMAIN ERROR
      RECIP←÷N←N-1
            ^
```

2.2 To borrow an idea from Z, we can use primes to indicate updated values. If function FOO updates a global variable X, we specify pre-conditions on that variable by referring to X, while post-conditions refer to X′. Thus we can clarify our earlier example, by writing

```
    RECIP←÷N′←N-1
```

and the old idiom

```
    S←(S≠' ')/S←⍞
```

becomes

```
    S″ ←(S′≠' ')/S′←⍞.
```

2.3 Phil Abrams once proposed that values should be bound at the start of execution of a line — this may have been in the days before diamond statement separators — which would have given us the sequence

```
    RECIP←÷N←N-1
```

where RECIP takes the value of the reciprocal of the value of N *before updating*.

In the event, the idea was never adopted, but it still has merit. In particular, it would allow us to rerun a bad line in the same environment, while debugging.

Languages like SQL have long had to cope with complicated updates which may ultimately have to be abandoned. The update is applied tentatively. If at any point the update fails to meet certain specified constraints, the transaction is rolled back, and the database left in its original state. If the update is completed satisfactorily, it is then committed explicitly, and the new data becomes visible to other users. This, of course, requires *transaction_start* and *transaction_end* markers. In the absence of such markers, updates are committed immediately, by default.

3. an improved mechanism

3.1 It is clear that current practice in APL is to commit updates immediately, by default, with no rollback option.

It is, however, helpful to think of all side-effects as tentative, until the end of the statement is reached. If the end of the statement is marked by a diamond or end-of-line, the side-effects are committed. This does not affect the current definition for ◇, which we would retain as the default semantics.

We would also introduce two new connectors, which are for the moment written as <&> and <•>.

At their simplest:

<&> would defer commitment until the end of the following statement;

<•> would trigger a rollback in the event of error in the left-hand statement, clear the error flag and execute the right-hand statement (which would presumably specify some sort of recovery routine) in the original environment. Otherwise, the interpreter would skip over the right hand statement to the next statement connector before deciding whether to commit outstanding updates.

3.2 For example,

```
      RECIP←÷N←N-1 <•> RECIP←0
```

would recover from an attempt to take the reciprocal of zero, but in doing so, it would also have rolled back the decrement of N, so that the more complete solution of

```
      RECIP←÷N←N-1 <•> RECIP←N←0
```

might be preferred.

Alternatively, the statement

```
      N←N-1 ◇ RECIP←÷N <•> RECIP←0
```

would ensure that the decrement is always committed, and the reciprocal operation is always protected.

3.3 The sequence

```
      THIS <&> THAT <&> THE_OTHER <•> RECOVER
```

defers commitment until THE_OTHER has completed execution. If this execution is error-free, the RECOVER statement is skipped. In the event of any error in the protected section, the interpreter rolls the environment back to the beginning of the line, undoing all updates made by THIS and THAT, before finally invoking RECOVER in the restored environment.

3.4 The sequence

```
    THIS <&> THAT <•> RECOVER ◇ THE_OTHER
```

defers commitment until THIS and THAT have completed execution. If this execution is error-free, the RECOVER statement is skipped, and interpretation continues with THE_OTHER. In the event of any error in the protected section, the interpreter rolls the environment back to the beginning of the line, undoing all updates made by THIS and THAT, before invoking first RECOVER and then THE_OTHER in the restored environment.

3.5 In the event that THIS executes without error, the sequence

```
    THIS <•> RECOVERY1 <•> RECOVERY2 ◇ THAT
```

would skip forward over RECOVERY1 and RECOVERY2, and execute THAT. In effect, the routine RECOVERY2 protects RECOVERY1.

3.6 If we are allowed to use braces to group lines together, the sequence

```
    {
      LINE1
      LINE2
      LINE3
    } <•> RECOVER
```

could (given some mechanism for passing back error flags) be made protect the whole block.

4. closing remarks

4.1 This is not the last word in error recovery, but it is an improvement.

To extend "error recovery" facilities into "event handling" facilities, the user program needs some means to indicate that it wishes either to abandon execution of the current statement, or to leave the current block with the error flag set, and ⎕signal or something similar would do the job very nicely.

4.2 Instead of returning to the most recently committed state, the programmer may prefer to return to some earlier environment. If that earlier environment falls within the current block then a branch command as the recovery statement may be all that is required.

If the desired earlier environment is external to the current block, then things get messy. It is a reasonable requirement, frequently needed in response to a "Quit" or "Cancel" command, but sometimes it can only be implemented using ugly constructs which trap error code 666 (or whatever) and pass back the error to the calling function, and so on, until the required environment is reached. This is all part of a larger problem, which may be tackled in due course.

4.3 The use of the tri-graphs, <&> and <•> , is highly unsatisfactory, but until better symbols suggest themselves, the chosen ones have the merit of suggesting the "and" and "or" element of the connectors' semantics.

4.4 A proper definition of the proposed new connectors is not itself a difficult task, but requires the definition of a new domain, $\mathbb{E}$, of executable statements.

Despite their central importance, executable statements are not very exciting objects: we define them (recursively), we combine them and we execute them.

The ways in which we define statements are defined by the syntax, the way they are executed is defined by the execution model, which only leaves the ways in which they are combined.

All diamond operators are elements of $\mathbb{E}\mapsfrom\langle\mathbb{E};\mathbb{E}\rangle$. The operators ◇ and <&> both deliver domain products, their results being members of $(\mathbb{E}\otimes\mathbb{E})$. The operator <•> , on the other hand, delivers a domain sum, the result being an element of $(\mathbb{E}\oplus\mathbb{E})$, but, since the construct

```
      THIS <•> RECOVERY1
```

is radically different from

```
      RECOVERY1 <•> THIS
```

it has the added complication that this particular domain sum is not commutative.

4.5 Anybody interested in pursuing this topic from the theoretical point of view might do worse than start with

Dana Scott (1971), *The Lattice of Flow Diagrams*
(in) E Engeler [ed], *Symposium on Semantics of Algorithmic Languages*
Springer-Verlag, Berlin – Heidelberg

which uses flow diagrams as a simple example to demonstrate the techniques of "approximation" and limit domains.

Dyalog.Net: APLScript and Things Textual

by John Daintree (email: johnd@dyadic.com)

Introduction

This is the second of three papers discussing Dyadic Systems' .Net compatible product, Dyalog.Net.

In this paper we discuss APLScript, a UNICODE based representation of APL programs.

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.

Why APLScript?

APL has traditionally been developed using dedicated development environments. These environments typically comprise of editors, debuggers, workspace explorer tools, and a host of other features that "enhance " the coding experience. In many cases, especially when one is writing a simple utility or application it may be more convenient to dispense with this complex environment.

APLScript is a version of Dyalog APL that allows us to write APL code in any UNICODE aware editor. Armed with just Notepad, or our favourite text editor, we can write and deploy our APL application.

Another advantage of using APLScript is that we can take advantage of existing source-control systems to save our APL code. Also, we can use standard text comparison utilities to quickly find changes across versions of our source code. In multiple language projects the APL source code can be saved alongside the source code of the other languages, and even viewed using the same tools.

Hello World

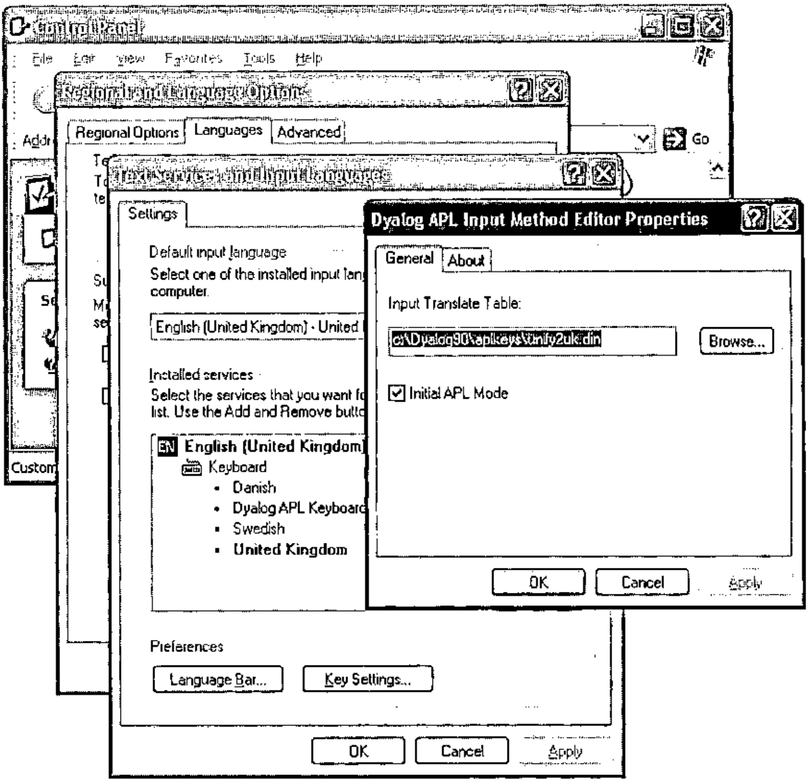
Using APLScript we can simply open Notepad and write a simple function:

```
∇hello
⎕←'Hello World'
∇
```

Typing APL into Notepad?

Dyalog.Net ships with a useful little utility called the APL IME. IME is an abbreviation of Input Method Editor. The APL IME allows us to type APL code (using our chosen APL translate table) into any application that is "IME aware".

The IME can be configured from the Control Panel



Dyalog APL Input Method Editor Properties

Input Translate Table: c:\Dyalog90\aplkeys\Unify2uk.din

☑ Initial APL Mode

Default input language

Select one of the installed input lan
computer.

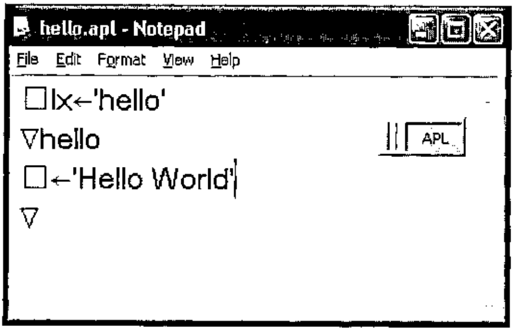
English (United Kingdom) - United

Installed services

Select the services that you want f
list. Use the Add and Remove butt

EN English (United Kingdom
- Keyboard
  - Danish
  - Dyalog APL Keyboard
  - Swedish
  - **United Kingdom**

During use, the APL IME presents itself as a small button that also indicates if APL input is currently enabled.



```
⎕lx←'hello'
∇hello
⎕←'Hello World'
∇
```

Back to the World

We can use the APLScript compiler to generate an executable file from this source code (which was saved in the text file hello.apl).

```
aplc /lx:hello /console hello.apl
```

The program aplc.exe is the APLScript compiler. This itself is written in Dyalog APL, and has been packaged into an executable by the Dyalog APL development environment.

On the command line we have used the /lx option to specify that the hello function is the "entry-point" of the program. The /console option indicates that this application can write to the host command window directly , and does not use the Dyalog session for input or output.

The full list of options for aplc.exe can be obtained with the command aplc /?, and this returns:

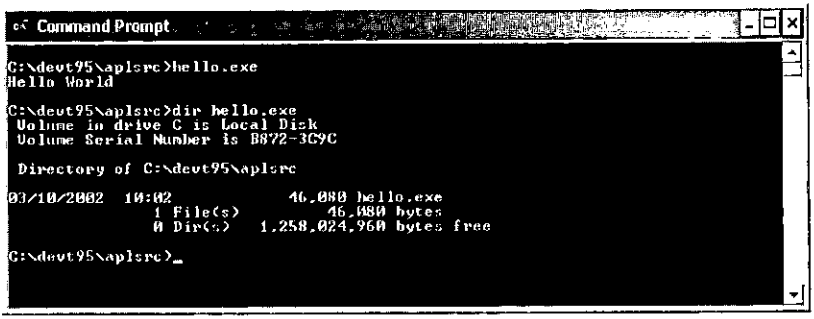
aplc.exe command line options:

/?	Usage
/r:file	Add reference to assembly
/o[ut]:file	Output file name
/x:file	Read source files from Visual Studio.Net project file
/res:file	Add resource to output file
/q	Operate quietly
/v	Verbose

/s	Treat warnings as errors
/runtime	Build a non-debuggable binary
/lx:expression	Specify entry point (Latent Expression)
/t:library	Build .Net library (.dll)
/t:nativeexe	Build native executable (.exe). Default
/t:workspace	Build dyalog workspace (.dws)
/nomessages	Process does not use windows messages. Use when creating a process to run under IIS
/console	Creates a console application

Some of these options will be discussed in this and future articles.

We can open a command window and execute hello.exe, which we can see is a "normal" executable file:



```
C:\devt95\aplsrc>hello.exe
Hello World

C:\devt95\aplsrc>dir hello.exe
 Volume in drive C is Local Disk
 Volume Serial Number is B872-3C9C

 Directory of C:\devt95\aplsrc

03/10/2002  10:02               46,080 hello.exe
               1 File(s)         46,080 bytes
               0 Dir(s)   1,258,024,960 bytes free

C:\devt95\aplsrc>_
```

This use of APLScript to create standalone executable files is independent of the .Net Framework. This example will compile and execute on any windows platform, with or without the .Net framework. The simplicity of the example illustrates the convenience of APLScript.

The Dyalog APL Development environment can be used to create executables directly from workspaces.

Writing Code in APLScript

The rules for APLScript are fairly simple. There are three things that can appear in APLScript.

1) An APLScript statement, e.g . `:Class, :EndClass, :Field`

2) A function definition, delimited by 'v'

3) An executable expression

APLScript Statements

:Class, :EndClass

e.g.

```
:Class Example:System.Object
      .
      .
      .
:EndClass
```

These APLScript declarations declare that the subsequent functions and code expressions comprise a new .Net Type. In full, we specify the name of the new class and the name of the base class. If the base class name is omitted it defaults to System.Object.

The example above declares a new Type called Example, that derives from System.Object.

:Field

e.g.

```
:Field Int32 Code
```

`:Field` defines a named "slot" within the class. This slot has a particular type and only data of the declared type can be stored in the field.

In the example we are declaring a field called `Code`, which can only contain data of type `Int32`

:Struct, :EndStruct

e.g.

```
:Struct Booking
    DateTime When
    String Customer
:EndStruct
```

These statements define a "structure". A structure is a set of name value pairs, which can be passed as a single entity to a function.

Each entry in a structure is similar to a field in that the data must be of the appropriate type.

:*Namespace*, :*EndNamespace*

e.g.

```
:Namespace utils
     .
     .
     .
:EndNamespace
```

Similar to :*Class*, but these declarations define a "traditional" namespace that will contain the subsequent functions and evaluated code expressions.

:*Property*, :*EndProperty*

e.g.

```
:Property Int32:IO
     ∇r←get
[1] r←⎕IO
∇
     ∇set value
[1] ⎕io←value
     ∇
:EndProperty
```

These statements allow us to define functions that will be called to set and get the values for a property.

This example above defines a property called *IO*.

The APLScript compiler generates a function called *get_IO* which contains the get portion of the code, and a *set_IO* function that contains the set portion.

Encapsulating access to ⎕*IO* as a property allows the calling language to use simple reference and assignment to access the current value. Using functions that perform the assignment and retrieval of the value allows the APL code to perform error checking on the value and throw appropriate exceptions if necessary.

A property can be made read-only by omitting the "set" portion of the :*Property* declaration.

`:Indexer, :EndIndexer`

e.g.

```
:Indexer Array:Item
:ParameterList Int32
    ∇set args
[1] value index←args
[2] value ⎕freplace tie index
    ∇
    ∇r←get index
[2] r←⎕fread tie index
    ∇
:EndIndexer
```

An *indexer* is a special type of property. The indexer specifies the "default" member of a Type. We will see another example of an indexer later.

APLScript Functions

e.g.

```
     ∇hello
[1]  ⎕←'Hello World'
     ∇
```

APL Functions in APLScript are indicated by the '∇' character.

Within the body of the function line numbers may be present but are ignored by the script compiler.

When a function appears within `:Class` and `:EndClass` statements additional statements may be contained within the body of the function. These statements determine the metadata for the function.

The permitted statements are as follows.

`:Access`

e.g.

```
      :Access Public
```

or

```
      :Access Constructor
```

or

```
        :Access ClassConstructor
```

or

```
        :Access Public Constructor
```

"*:Access Public*" indicates that the function is "exported" from the Type in which it is defined. A function can be marked as *Public* (the default) or *Private*. A private function can only be called by the APL code in the workspace, not by any code that instantiates an instance of the Type.

"*:Access Constructor*" indicates that the function is a constructor for the type. A constructor function is called on a new instance of the type and allows the APL code to initialise the contents of the new object.

"*:Access ClassConstructor*" indicates that the function is a class constructor for the type. This function will be called before any instances of the type are created. This allows the APL code to initialise any elements of the class that will be common to all instances.

As shown in the final example, qualifiers to *:Access* can be merged in a single *:Access* statement.

The *Constructor* and *ClassConstructor* qualifiers are only applicable within *:Class*/*:EndClass* statements

:ParameterList

e.g.

```
   :ParameterList Int32,String
```

or

```
   :ParameterList Int32 value,String forename,String surname
```

:ParameterList specifies the parameter types of a function. It is a comma separated list. Each element of the list specifies the type of the corresponding parameter. A name for the parameter can be specified, and this appears in the metadata, but is otherwise unused.

:ParameterList has no impact on how the function is called from APL, it is only used to provide metadata for the assembly that is created from the script.

:Returns

e.g.

```
   :Returns Int32
```

:Returns specifies the type of the return value for the function. If *:Returns* is omitted from a function definition then the function should not return a result.

:Implements

e.g.

```
   :Implements Iota
```

or

```
   :Implements PropGet IO
```

or

```
   :Implements PropSet IO
```

:Implements allows us to export the containing function with a name other than the name of the function itself . This allows us to specify that a function provides an "overload" of a method. In addition *:Implements* can be used to indicate that a function is a get or set *accessor* for a property.

Each of these statements provide the same information in APLScript that was provided by the various dialog boxes in the Dyalog.Net development environment.

APLScript Executable Expressions

Executable expressions are those expressions that appear outside of the body of a function.

Executable expressions are evaluated by the APLScript compiler **at compile time.** For example, consider the following APLScript program

```
⎕lx←'hello'
⎕cy 'DISPLAY'

toupper←{
      idx←⎕AV⍳⍵
      off←48×(idx≥18)∧idx≤43
      ⎕AV[idx+off]
 }

∇hello
[1] ⎕←DISPLAY toupper 'Hello World'∇
```

The first eight lines of the code are "executable expressions". As such they are executed at compile time. The assignment to ⎕*LX* prevents the error that the script compiler would generate if we do not used the /lx command line option. The *DISPLAY* workspace is copied into this executable at compile time, so there are no additional workspaces to ship with our executable. We can also fix dynamic functions in this section of an APLScript program, and these can be called as normal from the body of the script.

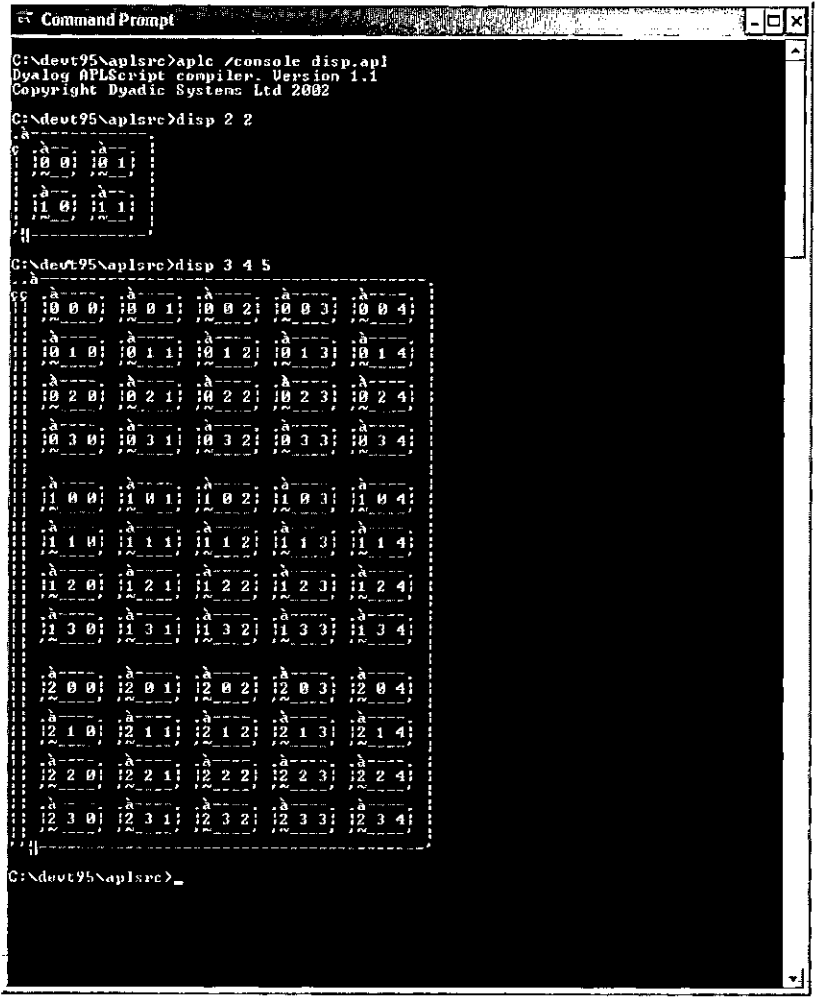
Reading the Command Line

Here is a simple APLScript program that uses the APL function *DISPLAY* to display the result of an APL primitive applied to the arguments passed to the program.

```
⎕lx←'disp'
⎕cy 'DISPLAY'


∇disp ;args
args←2 ⎕nq '.' 'GetCommandLineArgs'
⎕←DISPLAY ⍳⍎¨1↓args
∇
```

Note the new method *GetCommandLineArgs* on the root object. This method returns a vector of character vectors, providing the command line arguments of the program. *Args*[⎕*io*] is the name of the executing program, and so in this example is discarded. In addition there is a method called *GetCommandLine* that returns a character vector containing the command line as it was typed.

So, compiling and executing this program gives:



```
C:\devt95\aplsrc>aplc /console disp.apl
Dyalog APLScript compiler. Version 1.1
Copyright Dyadic Systems Ltd 2002

C:\devt95\aplsrc>disp 2 2
.à-------------.
ç .à--. .à--.  ¦
¦ ¦0 0¦ ¦0 1¦  ¦
¦ '~--' '~--'  ¦
¦              ¦
¦ .à--. .à--.  ¦
¦ ¦1 0¦ ¦1 1¦  ¦
¦ '~--' '~--'  ¦
'¦-------------'

C:\devt95\aplsrc>disp 3 4 5
..à-----------------------------------------------.
çç .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦0 0 0¦ ¦0 0 1¦ ¦0 0 2¦ ¦0 0 3¦ ¦0 0 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦0 1 0¦ ¦0 1 1¦ ¦0 1 2¦ ¦0 1 3¦ ¦0 1 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦0 2 0¦ ¦0 2 1¦ ¦0 2 2¦ ¦0 2 3¦ ¦0 2 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦0 3 0¦ ¦0 3 1¦ ¦0 3 2¦ ¦0 3 3¦ ¦0 3 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦                                          ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦1 0 0¦ ¦1 0 1¦ ¦1 0 2¦ ¦1 0 3¦ ¦1 0 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦1 1 0¦ ¦1 1 1¦ ¦1 1 2¦ ¦1 1 3¦ ¦1 1 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦1 2 0¦ ¦1 2 1¦ ¦1 2 2¦ ¦1 2 3¦ ¦1 2 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦1 3 0¦ ¦1 3 1¦ ¦1 3 2¦ ¦1 3 3¦ ¦1 3 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦                                          ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦2 0 0¦ ¦2 0 1¦ ¦2 0 2¦ ¦2 0 3¦ ¦2 0 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦2 1 0¦ ¦2 1 1¦ ¦2 1 2¦ ¦2 1 3¦ ¦2 1 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦2 2 0¦ ¦2 2 1¦ ¦2 2 2¦ ¦2 2 3¦ ¦2 2 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
¦¦ .à----. .à----. .à----. .à----. .à----.  ¦
¦¦ ¦2 3 0¦ ¦2 3 1¦ ¦2 3 2¦ ¦2 3 3¦ ¦2 3 4¦  ¦
¦¦ '~----' '~----' '~----' '~----' '~----'  ¦
''¦-----------------------------------------'

C:\devt95\aplsrc>_
```

Unfortunately the command window is unable to display the APL characters used by the *DISPLAY* function correctly.

APLScript and .Net Assemblies

In the first of these Dyalog.Net articles we used the Dyalog APL development environment to create a simple .Net Assembly that contained APL functions that could be called from a C# program. Let us now examine the APLScript version of the APL code from that article:

```
:Class Demo

    ∇make io
    :Access Constructor
    :ParameterList Int32

    ⍒io←io
    ∇

    ∇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,Int32
    :Returns Int32[][,]
    :Implements Method iota

    r←iota n
    ∇
:EndClass
```

Let's examine the code a section at a time

```
:Class Demo
```

We declare a new .Net Type called `Demo`. The base class is omitted so the new Type will derive from System.Object.

```
∇make io
:Access Constructor
:ParameterList Int32

⎕io←io
∇
```

The integer parameter of the constructor is used as the value of ⎕IO for this object.

Note that each instance of a .Net Type written in APL is its own namespace. This means that each instance can have its own global variables (inside the namespace) and because ⎕IO has "namespace scope" the assignment above will only affect this instance of the Demo Type. At any time there may be multiple instances of Demo, possibly with different values of ⎕IO.

```
∇r←iota n
```

Here we define a function called iota.

```
:Access Public
:ParameterList Int32[]
:Returns Array
```

The function is "public", which means that it can be called directly by code outside of the workspace.

The function is declared as taking a vector of integers, and returning an array of arbitrary rank and shape.

This is the most "generic" declaration that we can make for an iota function, and most completely describes the apl iota primitive.

```
r←⍳n
∇
```

The function just returns the result of the application of the APL primitive to the argument.

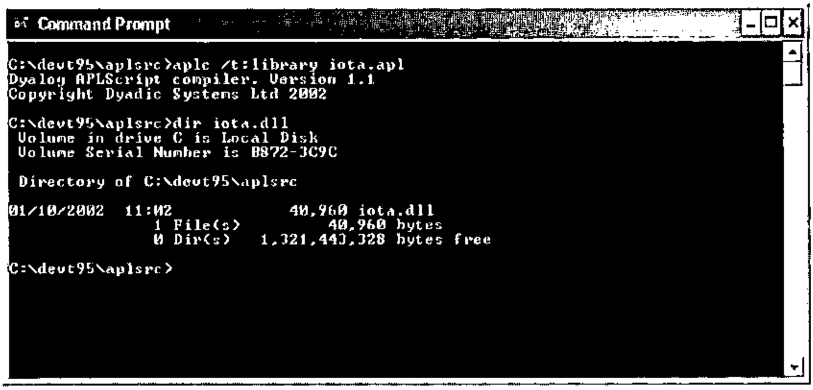
```
∇r←iota_1 n
:Access Public
:ParameterList Int32
:Returns Int32[]
:Implements Method iota

r←iota n
∇
```

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

r←iota n
∇
```

In addition to the basic `iota` function, we can also define "overloads" of `iota`. This is illustrated in the two functions above, each of which define an overload of `iota` and return arrays of rank 1 and 2 respectively. The `:Implements` statement declares that the methods `iota_1` and `iota_2` are visible externally with the name "iota". Note that `iota_1` and `iota_2` each call the previously declared `iota` function. `iota` is called as if it were any other APL function.

The definition of the class is terminated with the `:EndClass` statement.

This APL source code is saved in the file iota.apl (remember that this is UNICODE text file). Again, we can use the APLScript compiler to compile the APL code into a .Net Assembly



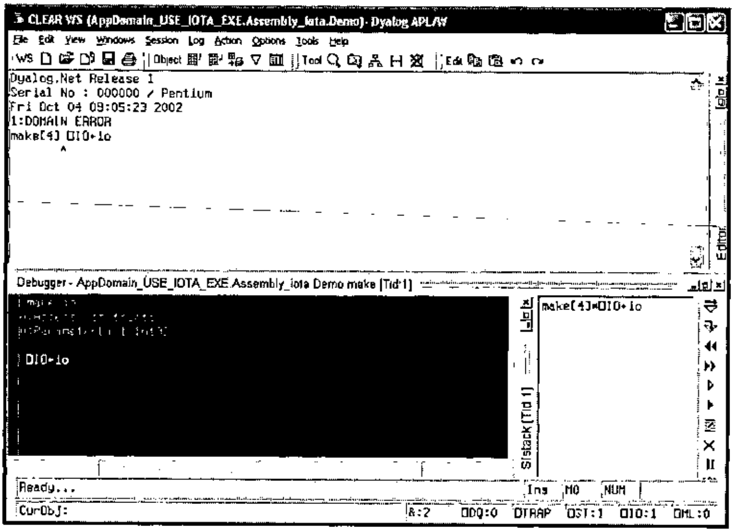
```
C:\devt95\aplsrc>aplc /t:library iota.apl
Dyalog APLScript compiler. Version 1.1
Copyright Dyadic Systems Ltd 2002

C:\devt95\aplsrc>dir iota.dll
 Volume in drive C is Local Disk
 Volume Serial Number is B872-3C9C

 Directory of C:\devt95\aplsrc

01/10/2002  11:02               40,960 iota.dll
               1 File(s)         40,960 bytes
               0 Dir(s)   1,321,443,328 bytes free

C:\devt95\aplsrc>
```

Similar C# code to that presented in the first article can then use the Demo Type contained within the assembly.

```
using System;

class UseIota
    {
    static void TestDemo()
        {
        Demo demo = new Demo(2);
        int[] shape={1,2,3};

        Console.WriteLine(demo.iota(10));
        Console.WriteLine(demo.iota(10,10));
        Console.WriteLine(demo.iota(shape));
        }

    static void Main()
        {
        TestDemo();
        }
    };
```

A Note about Debugging

If we have an error in our APL code then the Dyalog APL development environment is invoked at runtime to allow us to debug the application. Consider the case where the C# programmer attempts to create an instance of Demo, passing an invalid value for ⎕*IO* to the constructor function. The Dyalog debugger displays as shown below:



CLEAR WS (AppDomain_USE_IOTA_EXE.Assembly_Iota.Demo) - Dyalog APL/W
File Edit View Windows Session Log Action Options Tools Help
WS Object Tool Edit
Editor

```
Dyalog.Net Release 1
Serial No : 000000 / Pentium
Fri Oct 04 09:05:23 2002
1:DOMAIN ERROR
make[4] ⎕IO←io
       ∧
```

Debugger - AppDomain_USE_IOTA_EXE.Assembly_iota Demo make [Tid 1]

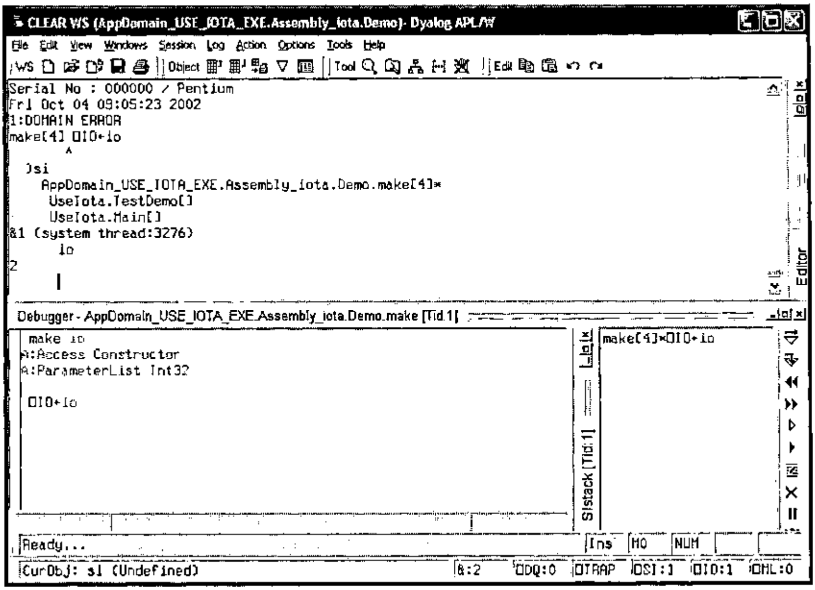
```
 ⎕IO←io
```

SIstack [Tid 1]

```
make[4]*⎕IO←io
```

```
Ready...                                          Ins   NO   NUM
CurObj:                              &:2   ⎕DQ:0   ⎕TRAP   ⎕SI:1   ⎕IO:1   ⎕ML:0
```

We can use the usual techniques to fix the problem. Notice that as shown below, *)SI* includes the full stack trace of the application, including the portion from the C# code. ⎕*SI* just returns the APL portion of the stack.


CLEAR WS (AppDomain_USE_IOTA_EXE.Assembly_iota.Demo)- Dyalog APL/W
File Edit View Windows Session Log Action Options Tools Help
WS Object Tool Edit
Editor
Debugger - AppDomain_USE_IOTA_EXE.Assembly_iota.Demo.make [Tid:1]
SIstack [Tid:1]

```
Serial No : 000000 / Pentium
Fri Oct 04 09:05:23 2002
1:DOMAIN ERROR
make[4] ⎕IO←io
       ∧
  )si
    AppDomain_USE_IOTA_EXE.Assembly_iota.Demo.make[4]*
     UseIota.TestDemo[]
     UseIota.Main[]
&1 (system thread:3276)
      io
2
```

```
 make io
⍝:Access Constructor
⍝:ParameterList Int32

 ⎕IO←io
```

```
make[4]*⎕IO←io
```

```
Ready...                                    Ins  NO  NUM
CurObj: si (Undefined)      &:2  ⎕DQ:0  ⎕TRAP  ⎕SI:1  ⎕IO:1  ⎕ML:0
```

The ideal solution to this error is to trap the error on assignment to ⎕*IO* and throw a .Net exception that the C# code can catch. The new constructor would look as follows

```
     ∇make io
     :Access Constructor
     :ParameterList Int32

     :Trap 0
     ⎕io←io
 :Else
     (ArgumentException.New 'io must be 0 or 1') ⎕signal 90
 :EndTrap
     ∇
```

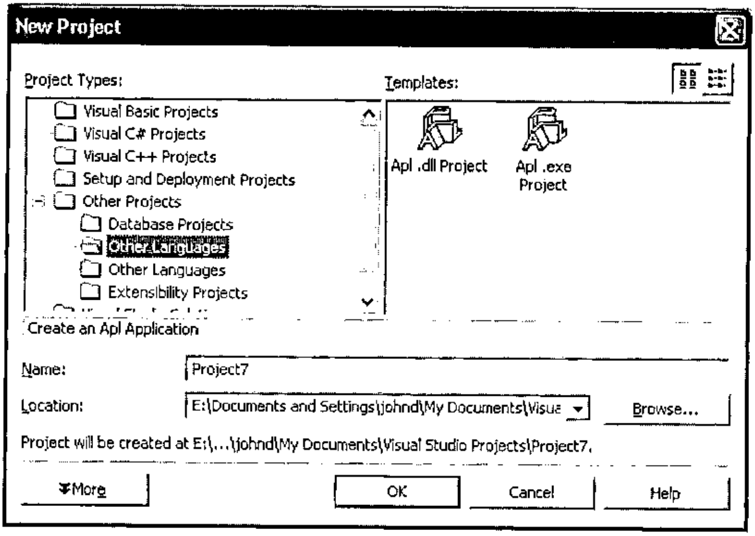
If we had compiled our APLScript library with the /runtime option then we would have seen the standard Dyalog "runtime violation" dialog box, not the debugger.

APLScript and Microsoft Visual Studio.

APLScript has allowed us to remove the APL language from the constraints of any particular development environment. We have seen that we can use Notepad, or any UNICODE text editor, to write APL code. In addition there is at least one important development environment that can now be used to develop APL code, and that is **Microsoft Visual Studio. Net.**

When Dyalog.Net is installed on a machine where Microsoft Visual Studio. Net (hereafter referred to as VS.NET) is installed, the installation process integrates APLScript with VS.NET.

When we select "New Project " from the VS.NET IDE we are given the option to create an "Apl exe project" or a "Apl dll project".



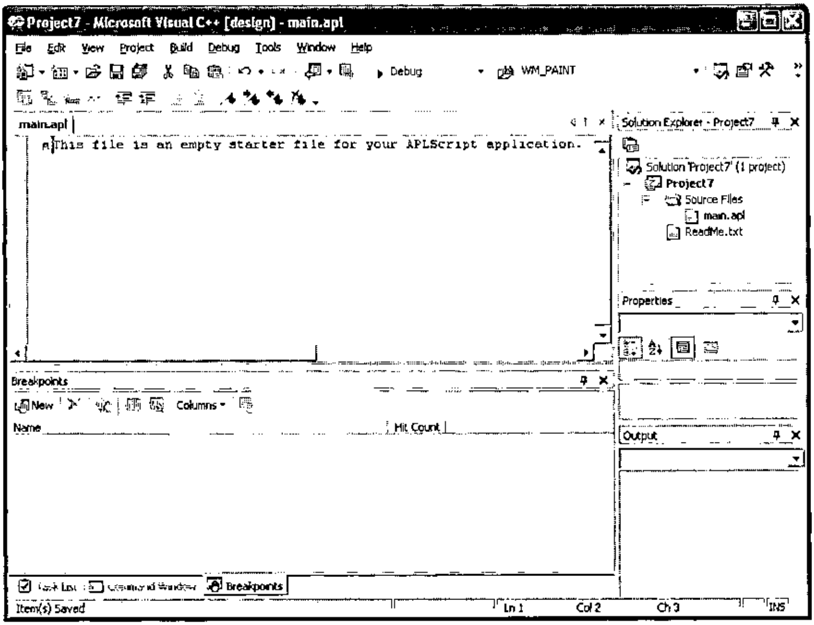
Create an Apl Application

Name: Project7

Location: E:\Documents and Settings\johnd\My Documents\Visua

Project will be created at E:\...\johnd\My Documents\Visual Studio Projects\Project7.

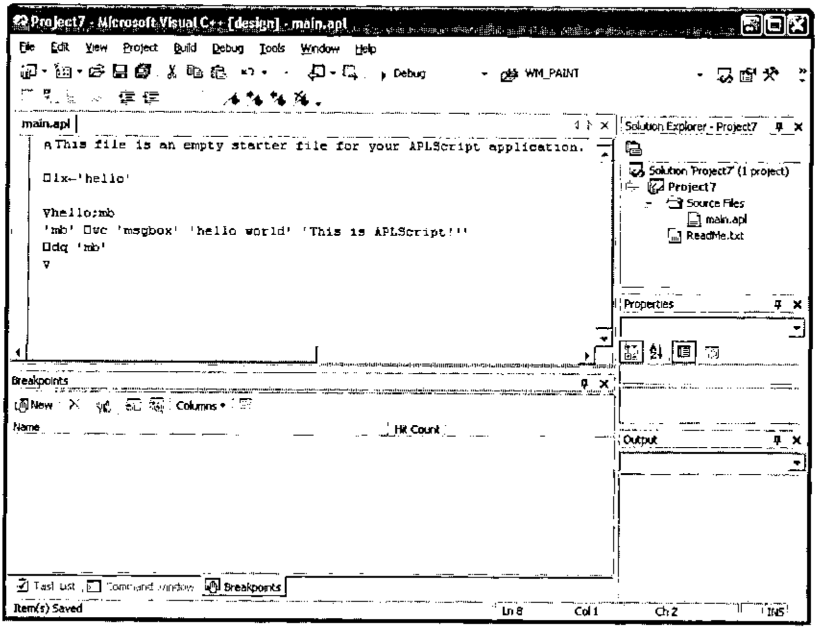
Selecting "Apl .exe project" causes VS.NET to create an empty project for us that initially contains a file called main.apl which is our initial APLScript source code document.



```
⍝This file is an empty starter file for your APLScript application.
```

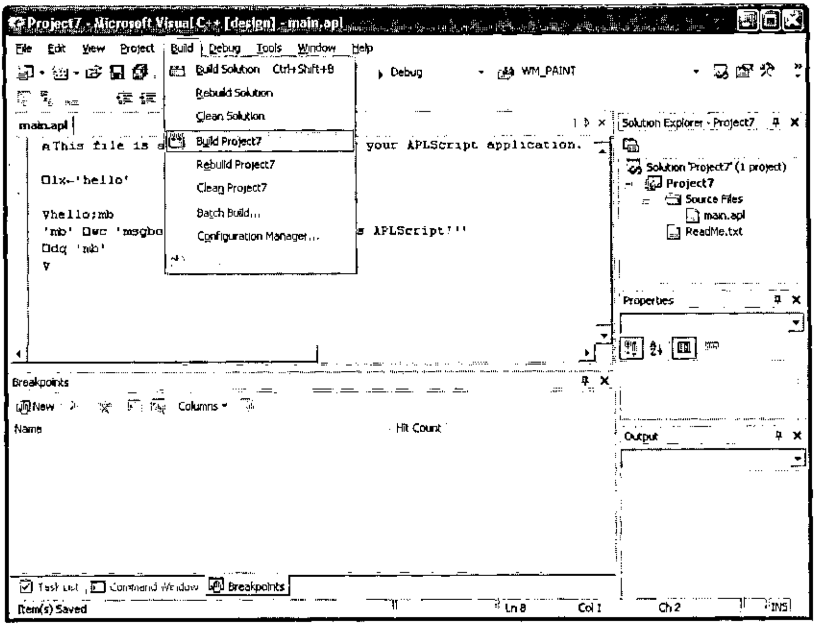
Using the APL IME we can type our APL code into the VS.NET editor.

Here is a simple GUI "hello world" program



```
⍝This file is an empty starter file for your APLScript application.

⎕lx←'hello'

∇hello;mb
'mb' ⎕wc 'msgbox' 'hello world' 'This is APLScript!''
⎕dq 'mb'
∇
```

We can select "Build Project" from the Build Menu

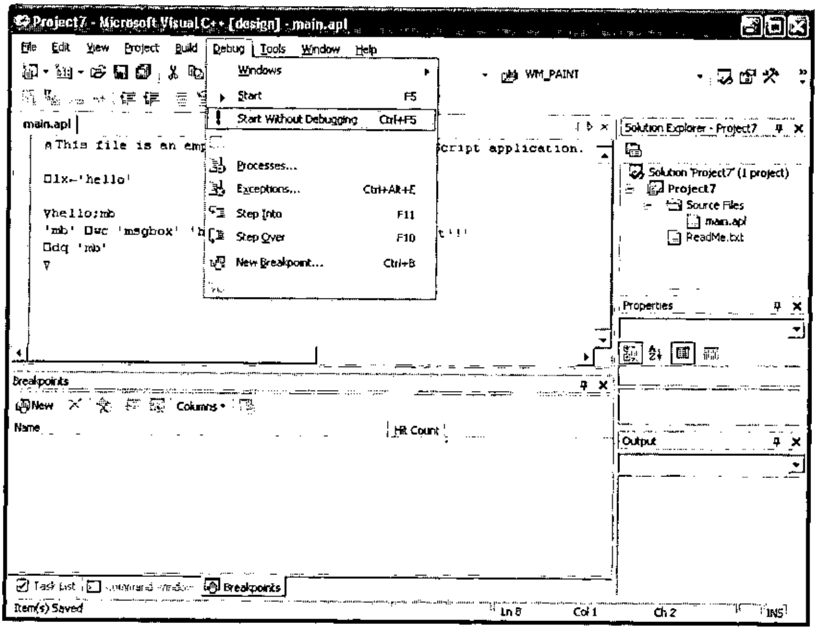


Project7 - Microsoft Visual C++ [design] - main.apl

```
⍝This file is a            your APLScript application.

⎕lx←'hello'

∇hello;mb
'mb' ⎕wc 'msgbo            s APLScript!'''
⎕dq 'mb'
∇
```

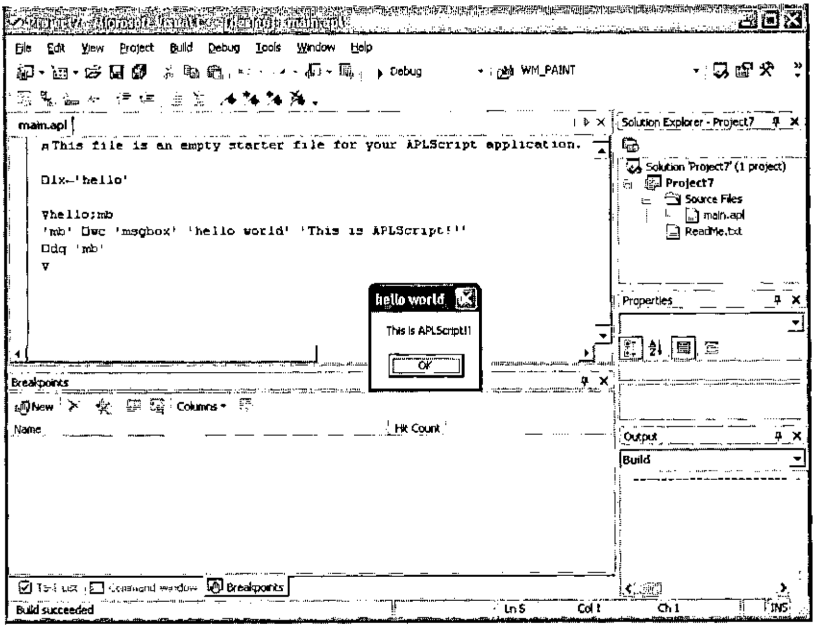
And this invokes the APLScript compiler, which compiles the source code.

Finally, we can execute project7.exe.



```
⍝This file is an emp                    Script application.

⎕lx←'hello'

∇hello;mb
'mb' ⎕wc 'msgbox' 'h                    t''!'
⎕dq 'mb'
∇
```

And we get…



main.apl

```
⍝This file is an empty starter file for your APLScript application.

⎕lx←'hello'

∇hello;mb
'mb' ⎕wc 'msgbox' 'hello world' 'This is APLScript!'
⎕dq 'mb'
∇
```

hello world

This is APLScript!!

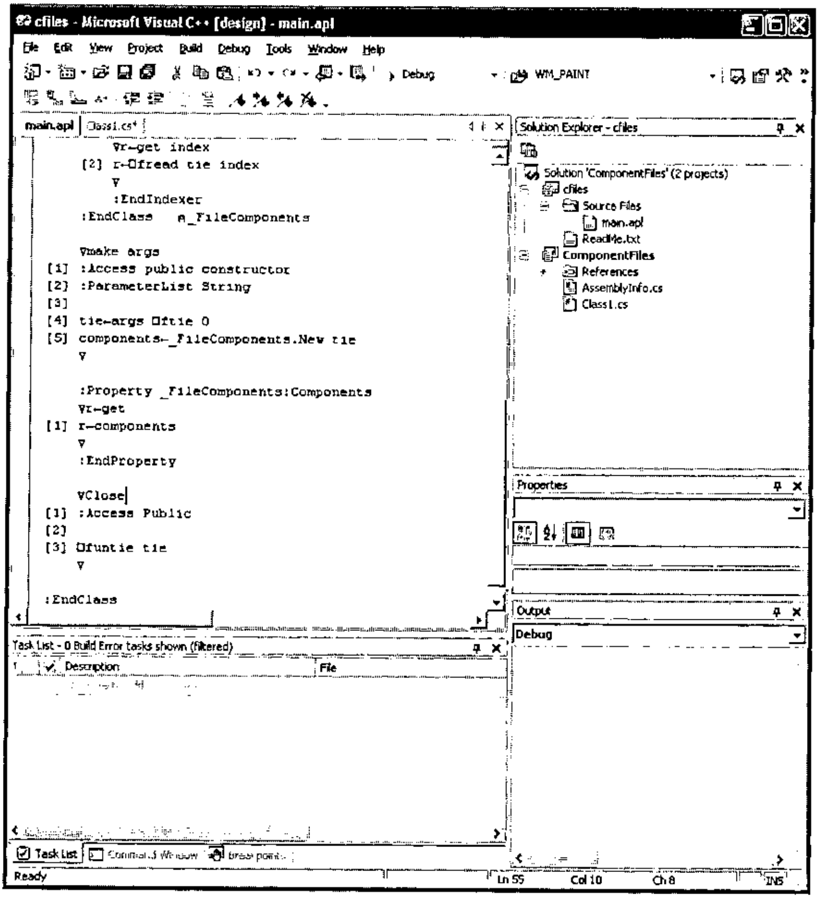
Admittedly a trivial example, but it indicates the portability of APLScript.

Let's look at a more complicated example that uses two programming languages, some .Net, and some APL component files.

The ComponentFile Solution

Dyalog.Net ships with a VS.NET *solution* called ComponentFiles. A solution is a number of related projects. In ComponentFiles we have a project called cfiles, which is written in APLScript, and a project called ComponentFiles which contains C# source code.

If we open the ComponentFiles solution in VS.NET we see the following:


cfiles - Microsoft Visual C++ [design] - main.apl
File Edit View Project Build Debug Tools Window Help
Debug
WM_PAINT
main.apl
Class1.cs*
Solution Explorer - cfiles
Solution 'ComponentFiles' (2 projects)
cfiles
Source Files
main.apl
ReadMe.txt
ComponentFiles
References
AssemblyInfo.cs
Class1.cs
Properties
Output
Debug
Task List - 0 Build Error tasks shown (filtered)
Description
File
Task List
Command Window
Breakpoints
Ready
Ln 55
Col 10
Ch 8
INS

```
        ∇r←get index
    [2] r←⎕fread tie index
        ∇
        :EndIndexer
    :EndClass   ⍝_FileComponents

    ∇make args
[1] :Access public constructor
[2] :ParameterList String
[3]
[4] tie←args ⎕ftie 0
[5] components←_FileComponents.New tie
    ∇

    :Property _FileComponents:Components
    ∇r←get
[1] r←components
    ∇
    :EndProperty

    ∇Close
[1] :Access Public
[2]
[3] ⎕funtie tie
    ∇

:EndClass
```

The ComponentFiles solution contains an APLscript program that presents an object-oriented interface to APL component files. Also in ComponentFiles is a C# program that uses the APL code to retrieve and modify components in a simple component file.

On the right hand side of the window we can see the “Solution Explorer” which lists the files that make up the solution, and we can see the current source file open in the editor. This is main.apl and is the APLScript code for the cfiles project.

Here is the full listing of the APLScript program

```
⍝ This file is an empty starter file for your APLScript application.
⎕io←1
⎕ml←0


:Class ComponentFile


    :Class _FileComponents
         ∇make arg
         :Access constructor
         :ParameterList Int32

         tie←arg
         ∇


         :Property Int32:Count
         ∇r←get
         r←¯1+2⊃⎕fsize tie
         ∇
         :EndProperty

         ∇r←Add array
         :Access public
         :ParameterList Array
         :Returns Int32

         r←array ⎕fappend tie
         ∇
```

```
          :Indexer Array:Item
          :ParameterList Int32
          ∇set args
      [1] (2⊃args) ⎕freplace tie (1⊃args)
          ∇
          ∇r←get index
      [2] r←⎕fread tie index
          ∇
          :EndIndexer
      :EndClass⍝ _FileComponents

      ∇make args
  [1] :Access public constructor
  [2] :ParameterList String
  [3]
  [4] tie←args ⎕ftie 0
  [5] components←_FileComponents.New tie
      ∇

      :Property _FileComponents:Components
      ∇r←get
  [1] r←components
      ∇
      :EndProperty

     ∇Close
  [1] :Access Public
  [2]
  [3] ⎕funtie tie
      ∇

  :EndClass
```

The APLScript defines two .Net types , one called `ComponentFile` and one called *`_FileComponents`*. The `ComponentFile` Type encompasses operations that can be performed on a file, e.g. *`Close`*. The *`_FileComponents`* Type encompasses operations that can be performed on components in the file, such as *`Add`* and *`Count`*. The *`_FileComponents`* Type uses an indexer to present the components in the file as an array. Notice also that *`_FileComponents`* is declared **inside** the definition of `ComponentFile`. This allows us to define, in .Net terminology, a **"Nested Type"**.

When an instance of `ComponentFile` is created the constructor function (`make`) is called with the filename of the desired component file as a parameter. The `make` function ties the component file and stores the tie number in a variable called `tie`. This variable is global to the current namespace and so is specific to this particular instance of the `ComponentFile` object, each instance of `ComponentFile` has its own namespace. In addition the make routine creates a new instance of the `_FileComponents` class, passing the new tie number to the constructor of this Type.

The `ComponentFile` object has a property called `Components` that returns the instance of the `_FileComponents` class that was created in the `ComponentFile` constructor. Note that this is a read-only property.

Here's the C# program that accesses the ComponentFile Type.

```
using System;

namespace ComponentFiles
{
      /// <summary>
      /// Summary description for Class1.
      /// </summary>
      class Class1
      {
            public static void Main()
            {
                  ComponentFile file = new ComponentFile(".\\cfiles.dcf");

                  for (int i=1;i<=file.Components.Count;i++)
                              DumpArray(file.Components[i]);

                  Console.WriteLine(file.ToString());

                  Console.Write("file.Count:");
                  Console.WriteLine(file.Components.Count);

                  Console.Write("file.Components[1]:");
                  DumpArray(file.Components[1]);

                  int[] New = new int[3];

                  New[0]=1;
                  New[1]=3;
                  New[2]=5;

                  Console.Write("Added component at ");
                  int at = file.Components.Add(New);

                  Console.WriteLine(at);

                  Console.Write("New component contains:");
                  DumpArray(file.Components[at]);

                  New[0]=11;
                  New[1]=33;
                  New[2]=55;

                  Console.Write("Overwritten component.Now contains:");
                  file.Components[at]=New;
                  DumpArray(file.Components[at]);

                  file.Close();
```

```
        }
        static void DumpArray(Array a)
        {
            switch (a.Rank)
            {
                case 1:
                    for (int i=0;i<a.Length;i++)
                    {
                        if (i!=0)
                            Console.Write(",");
                        Console.Write(a.GetValue(i));
                    }
                    break;
                default:
                        Console.Write(a.ToString());
                        break;
            }
            Console.WriteLine();
        }
    }
}
```

Look at the following line of the C# program

```
    Console.WriteLine(file.Components.Count);
```

The variable `file` refers to an instance of the *ComponentFile* Type. *ComponentFile* has a property, *Components*, that returns an instance of the *_FileComponents* Type, which has a *Count* property. When this line of C# is executed the code specified in the get section of the *:Property* declaration of *Count* is executed.

Remember that _FileComponents defined an indexer called Item:

```
      :Indexer Array:Item
      :ParameterList Int32
      ∇set args
  [1] (2⊃args) ⎕freplace tie (1⊃args)
      ∇
      ∇r←get index
  [2] r←⎕fread tie index
      ∇
      :EndIndexer
```

This indicates that the _FileComponent Type has a Property called Item, that is the *default* member of the class. If Item was declared using :Property the C# code would have to use

```
array=File.Components.Item[2] ;
```

To access the 2nd component of the file. Because we used :Indexer to declare Item, the C# code can be simplified to

```
array=File.Components[2];
```

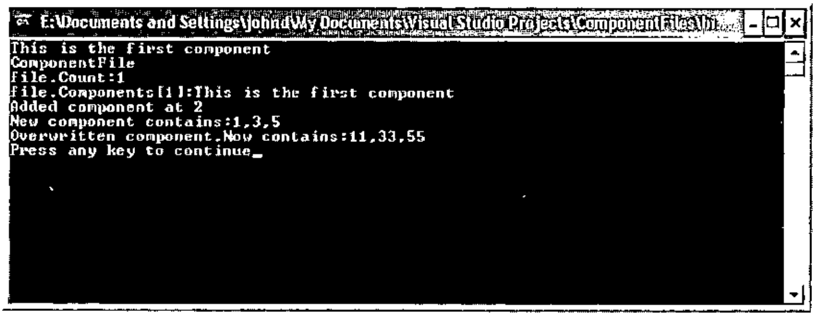
which is somewhat more convenient for the C# programmer. The APL code that is executed in each case would be identical.

Similarly, assignment to the appropriate element of the component file can be achieved with

```
File.Components[2]=array;
```

And this will execute the APL code in the set section of the :Indexer declaration.

If we initialize the component file so that it contains a single component that is the vector 'This is the first component' the output from the C# program is as follows


E:\Documents and Settings\johnd\My Documents\Visual Studio Projects\ComponentFiles\bi

```
This is the first component
ComponentFile
File.Count:1
File.Components[1]:This is the first component
Added component at 2
New component contains:1,3,5
Overwritten component.Now contains:11,33,55
Press any key to continue_
```

Summary

APLScript allows us to abstract the APL language from the traditional developments, giving us the ability integrate our APL code with other languages and development environments.

The Next Article

In the next article we will examine the use of APLScript to write web pages.

John Daintree
Senior Programmer
Dyadic Systems Ltd

email: johnd@dyadic.com
mailing list: dotnet@dyadic.com

Index to Advertisers

Dyadic Systems Ltd	2
MicroAPL Ltd	8
Strand Software	11
Vector Back Numbers	5

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

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

Signature: ______________________________

Send the completed form to:
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.

2002/2003 Committee

Chairman	Adrian Smith 01439-788385 apl385@compuserve.com	Brook House Gilling East YORK YO62 4JJ
Secretary	Anthony Camacho 0117-973-0036 acam@tesco.net	11 Auburn Road Redland BRISTOL, BS6 6LS
Treasurer	Nicholas Small 01223-570850 treas.apl@bcs.org.uk	12 Cambridge Road, Waterbeach, Cambridge CB5 9NJ
Journal Editor	Stefano Lanzavecchia stf@apl.it	c/o APL Italiana Corso Vercelli 54 20145 - Milano Italy
Projects and Publicity	Dr Alan Mayer 01792-205678x4274 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Webmaster	Ray Cannon 01252-874697 ray_cannon@compuserve.com	21 Woodbridge Road, Blackwater, Camberley, Surrey GU17 0BS
Activities	Jon Sandles 01904-612882 jon_sandles@csi.com	138 Burton Stone Lane, YORK YO30 6DF
School Project	Stephen Taylor sjt@lambenttechnology.com	
Administration	Rowena Small 01223-570850 treas.apl@bcs.org.uk	12 Cambridge Road, Waterbeach, Cambridge CB5 9NJ

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

VECTOR is the quarterly Journal of the British APL Association and is distributed to Association members in the UK and overseas. The British APL Association is a Specialist Group of the British Computer Society. APL stands for "A Programming Language" – an interactive computer language noted for its elegance, conciseness and fast development speed. It is supported on most mainframes, workstations and personal computers.

SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

Causeway Graphical Systems Ltd
The Maltings, Castlegate,
MALTON, North Yorks YO17 7DP
Tel: 01653-696760
Fax: 01653-697719
Email: sales@causeway.co.uk
Web: www.causeway.co.uk

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

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

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

APL2000 Inc
One Research Court
Suite 140
Rockville MD 20850
USA
Tel: +1 (301) 208 7150
Email: sales@apl2000.com
Web: www.apl2000.com

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

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel: +31 347 342 337

HMW Computing
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel: 0870-1010-469
Email: HMW@4xtra.com

Soliton Associates Ltd
Groot Blankenberg 53
1082 AC Amsterdam
Netherlands
Tel: +31 20 646 4475
Fax: +31 20 644 1206
Email: sales@soliton.com