Contents

Editorial:

Discover, Learn, Profit

The BAA's mission is to promote the use of the APLs. This has two parts: to serve APL programmers, and to introduce APL to others.

Printing *Vector* for the last twenty years has addressed the first part, though I see plenty more to do for working programmers – we publish so little on APL2, for example. But it does little to introduce APL to outsiders.

If you've used the Web to explore a new programming language in the last few years, you will have had an experience hard to match with APL. Languages and technologies such as JavaScript, PHP, Ruby, ASP.NET and MySQL offer online reference material, tutorials and user forums. In our world, Jsoftware has for some time had the best and most active of these but until the recent appearance of its excellent new site (*http://www.jsoftware.com*), wiki and forums, little could be viewed or googled through the Web. (Congratulations to Eric Iverson and Chris Burke for an excellent production.)

Vector now gives priority to publishing online. This is the first issue to appear simultaneously in print and online. We are steadily bringing our twenty-year archive on line.

This issue is also the first to follow the structure of Vector Online. In place of the division between 'technical' and 'general' articles, we offer sections Discover, Learn and Profit.

- **Discover** is about the APLs, possible extensions to them, their history and their relationships with other programming languages;

- **Learn** is about extending our competence, for new and experienced programmers both;

- **Profit** is about profiting from the use of APLs, either commercially, academically, or simply in pursuit of a hobby.

Gene McDonnell's "At Play With J", which would usually appear in Learn, appears this issue in Discover, as Gene reports the results of a programming competition. Successful entries for the competition ran up to 100 lines long; Metlov's J entry runs to 6 characters. Hello?

Sudoku madness Do programmers solve Sudoku problems – or just write programs that do? The Sudoku craze has provoked our readers to offer general solutions; two in Dyalog APL and a characteristically terse J program from Roger Hui. We doubt even their authors intended to profit from them by using them to solve Sudoku problems; so we're offering them as examples to learn from. We can all profit by studying them.

Stephen Taylor
email: editor@vector.org.uk

Dates for Future Issues of VECTOR

	Vol.22 No.1	Vol.22 No.2	Vol.22 No.3
Copy date	in press	3rd Dec	3rd March
Ad booking	in press	10th Dec	10th March
Ad Copy	in press	17th Dec	17th March
Distribution	Dec 2005	Jan 2006	April 2006

```
Back numbers advert (rerun)
```

RESPONSES

Thoughts on the Urn Problem

from Eddie Clough (eacloughatt@tiscali.com)

In his article in Vector Vol. 21 No. 2, Devon McCormick purports to show how, given an urn containing a known number of balls, each of which may be black or white, Bayesian statistics can be used to derive the probability that all of the balls are white after a number of white and only white balls have been drawn, with replacement. No one denies the validity of the Bayes formula, of course, but I am not convinced that it can be applied to the urn problem in the way McCormick suggests.

To understand the problem better, I asked myself how I would provide a prior probability. Since I have the opportunity to examine individual balls, I should provide a probability p that any particular ball is white. It will then follow that the probability of there being x white balls among n balls is given by the usual binomial formula (using the J vocabulary):

    (x!n)*(p^x)*(1-p)^(n-x)

Note that the value p is to be my Bayesian prior relating to the balls in a particular urn and in this context the use of the binomial formula is not itself a Bayesian prior.

{It is normally taken that the urn is a 'one-off'. There is no distribution of numbers of white balls involved so discussion of whether it is or is not binomial is not meaningful.}

The Bayes formula can then be used to determine a probability, say *P3*, that all of 5 balls are white after 3 three draws. My J4.06 script is:

```
ab=:3 :'((y.%n)^d)*(y.!n)*(py^y.)*(1-py)^(n-y.)'      NB. Bayes component
urn=:4 :0                        NB. e.g. (d,n) urn p [ ('d';'n';'p')=.3;5;0.5
d=:{.x. [   n=:{:x. [   py=:y.
(ab n)%+/ab"0 i.>:n                                    NB. Bayes probability
)
```

Graph 1 shows the relationship between p and $P3$ over the possible range of p.


GRAPH 1
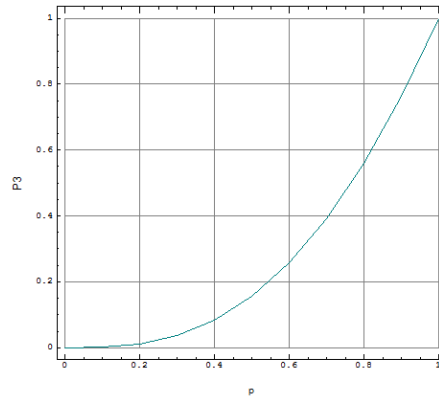
Choosing a value for p is the difficult bit. In general a person's choice will depend on whether they are optimistic ("I am sure they will all be white balls") or pessimistic ("I never win anything"), trusting ("Urn suppliers are good chaps") or suspicious ("There is always one bad apple"), and also, I believe, the person will be more or less strongly influenced by the intrinsic value of making a wrong or right decision about the urn – involving assessment against their own particular utility curve ("From my point of view, the stakes are high, so I will act with caution").

I am not quite sure where the implicit assumption is made but Devon McCormick's result of a P-value of 0.15625 corresponds to a p-value of 0.5. This may not seem as plausible a 'neutral' prior as does selecting the binomial distribution as the prior; it is equivalent to assuming, before any draws have been made, that the (prior) probability of all balls being white is P0= 0.03125 (calculated by `0 5 urn 0.5`).

An alternative start point might be to choose, as the prior, a probability of P0= 0.5 of all the balls being white. By trial and error or by more formal iterative means, it can be found that this corresponds to a probability p=0.8705507 of individual balls being white. This in turn implies a probability P3=0.699031 of all balls being white after three whites have been drawn. Is this a better solution? Is there a best neutral prior?

The relationship between P0 and P3 is shown in Graph 2, J for which is:

```
P0=:0 5 urn"1 0 p=:(10%~i.11)
P3=:3 5 urn"1 0 p
load'plot'
'frame 1;grids 1;title GRAPH 2;xcaption P0;ycaption P3' plot P0;P3
```

GRAPH 2
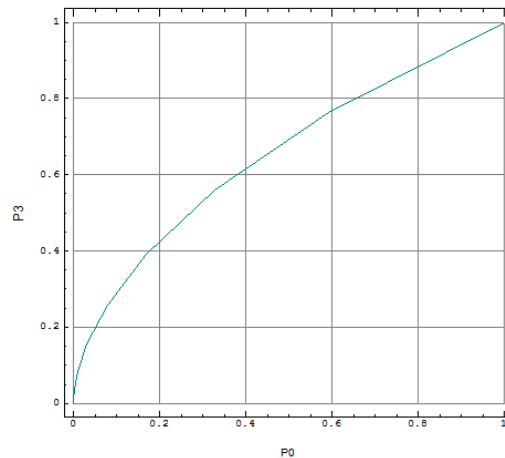
It can be seen more clearly that in reality there is no satisfactory neutral starting point if the variable `p` is transformed. Probability occupies the domain (0,1). If I choose to work with odds, `z=p%(1-p)`, rather than probability so that the binomial formula becomes

```
(x!n)*(z^x)%(1+z)^n
```

(a slightly simpler formula than the original in that `x` only appears twice), I then have a variable, which occupies the domain (`0,_`), i.e. zero to infinity. Finally, taking log odds I finish with a variable in the domain (`__,_`) or minus to plus infinity. In this more familiar territory it can be recognised that at least two parameters, a mean and a variance, are needed to describe my initial state of mind and in fact the variance is the dominant parameter, for, if there is no prior information about `p`, the appropriate value for the variance of log `z` is infinity and other parameters become indeterminate.

My conclusion is that the Bayes formula can and should be used to show the logical relationship between the various probabilities involved in a problem but if the process is taken beyond this point it may only offer fool's gold. Perhaps those who came after Laplace, to whom Devon McCormick refers, were more astute than we think.

A person might still select a prior – either p or P0 – to reflect his own views but I would suggest that, if the number of draws is less than the number of balls, the best advice a statistician can give is a conditional probability of the form:
If there is one black ball the probability of not drawing it in three draws is `(4%5)^3=0.512`, hence there can be no great confidence that all the balls are white.

When the number of draws is equal to or greater than the number of balls, a different question can be examined, namely what is the probability that all the balls have been seen, and if only white balls have been drawn this can be used as a measure of the probability that all the balls are white. Such a probability can be computed without reference to a prior. The rows in the following table give the probabilities for urns with different numbers of balls.

```
+--+--------------------------------------------------------------+
|  |                       Number of Draws                        |
+--+--------------------------------------------------------------+
|  |    2            3            5            10           20     |
+--+--------------------------------------------------------------+
|  2|  5.000e_1     7.500e_1     9.375e_1     9.980e_1     1.000e0 |
|  3|  0.000e0      2.222e_1     6.173e_1     9.480e_1     9.991e_1 |
|  5|  0.000e0      0.000e0      3.840e_2     5.225e_1     9.427e_1 |
|10|  0.000e0      0.000e0      0.000e0      3.629e_4     2.147e_1 |
+--+--------------------------------------------------------------+
```

CRCOMPARE in J

from Adam Dunne (Tamil Nadu, India)

The attached code (written in J but without any tacit coding for simplicity) is shorter than the APL version in *Vector*, but perhaps more importantly, I think the code is easier to grasp, resting as it does on one central idea, a 'matching lines' table, shown below (each line of `cr1` is a row, each line of `cr2` is a column; matches are flagged). One can intuitively see how the code extracts the line numbers from the table indices. A blank line is added to the top of `cr1` and `cr2` in `crcreate` as a 1 in position (0,0) of the table is always needed to make it work.

J does not use line numbers, but I thought they might be useful in an application like `crcompare`, so I added them. Sample output follows ...

```
'cr1 cr2'=.'calc'crcreate'calc2'
   addlinenos cr1
 0|
 1|3 : 0
 2|display a+b+c
 3|d=.3
 4|e=.d=.a+b+c
 5|f=.d+e
 6|a=.0 0 $0
 7|b=.0 0$0
 8|g=.f^0.5
 9|display'done'
10|z=.0.01*d
```

```
  11|)

    addlinenos cr2
   0|
   1|3 : 0
   2|display a+b+c]d=.3
   3|e=.d=.a+b+c
   4|f=.d+e
   5|g=.f^0.5
   6|a=.b=.0 0$0
   7|display'done'
   8|z*d%100
   9|)
```

```
    NB. This is the table of matching lines
    cr1-:/"1 1"1 2 cr2
1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1
```

```
    'calc'crcompare'calc2'
   2|display a+b+c
   3|d=.3
-replaced by----
   2|display a+b+c]d=.3

   6|a=.0 0 $0
   7|b=.0 0$0
----deleted----

   6|a=.b=.0 0$0
----added----

  10|z=.0.01*d
-replaced by----
   8|z*d%100
```

```
display=:(1!:2)&2

crcompare=: 4 : 0
'cr1 cr2'=.x.crcreate y.
cr1nos=.addlinenos cr1
cr2nos=.addlinenos cr2
linesmatchtab=.cr1-:/"1 1"1 2 cr2
inds=.I.1=,linesmatchtab
ind2=.($linesmatchtab)#:inds
z=.0 0,<:(}.ind2)-}:ind2 NB. indices increment
z2=.ind2-z              NB. starting indices
z3=.i.0
for_ct.i.{.$z do.
```

```
   z3=.z3,(((<ct,0){z2)+i.(<ct,0){z);((<ct,1){z2)+i.(<ct,1){z
end.
z4=.((-:$z3),2)$z3
lineflags=.dims"0 z4
zout=.0 0$0
width=.1{$cr1
for_ct.i.{.$z4 do.
  flagsrow=.ct{lineflags
  'extralines1 extralines2'=.ct{z4
  if.flagsrow-:1;1 do.      NB. replace
    zout=.zout,(extralines1{cr1nos),(width{.'-replaced by----')
    zout=.zout,(extralines2{cr2nos),(1,width)$' '
  elseif.flagsrow-:1;0 do. NB. deleted
    zout=.zout,(extralines1{cr1nos),(width{.'----deleted----'),(1,width)$' '
  elseif.flagsrow-:0;1 do. NB. added
    zout=.zout,(extralines2{cr2nos),(width{.'----added----'),(1,width)$' '
  end.
end.
zout=.charmattovec zout
)


samewidth=: 4 :'(maxw{."1 x.);(maxw=.>./(1{$x.),1{$y.){."1 y.'
charvectomat=: 3 :',;._2 y.,LF'NB.converts char vec with LFs to char mat
crcreate=:4 :'('' '',charvectomat 5!:5<x.)samewidth '' '',charvectomat
5!:5<y.'
dims=:3 :'<(#>y.)>0' NB. flags boxes where dimension>0
elim_trail_bl=:3 :'(($y.)-(|.'' ''=y.)i.0){.y.'
NB.eliminates trailing blanks
charmattovec=:3 :0 NB. converts character matrix to char vector with LFs
z=.i.0
for_ct.i.{.$y. do.
  z=.(z,elim_trail_bl ct{y.),LF
end.
)


addlinenos=:3 :'(3":,.i.{.$y.),.''|'',.y.' NB. adds linenos to crfn
```

News from Sustaining Members

MicroAPL Ltd

The launch by AMD of a 64-bit version of the x86 architecture, a couple of years ago, has opened the way for low-cost, 64-bit systems in the mass market. Today, you can buy a desktop machine – or even a laptop – with a 64-bit processor, for just a few hundred pounds. Intel have now followed AMD with a binary-compatible equivalent, and, over the next few months, we can expect 64-bit x86 systems to become commonplace. 64-bit versions of Windows and Linux are available to run on this architecture. In addition, Apple has sold 64-bit Macintoshes, based on the PowerPC architecture, for some time. 64-bit Servers are, of course, well established.

Such machines can address extraordinary amounts of memory. 32-bit systems are limited to at most 4Gb of memory address space, and usually this is not all available to user processes (Windows, for example, normally has a limit of 2Gb). Today, XP Professional x64 Edition supports up to 128Gb of RAM and 16 Terabytes of virtual memory. In the future, even more memory will be usable. And the cost of RAM has fallen to such an extent that it is already possible to configure desktop machines with tens of gigabytes of memory for a couple of thousand pounds.

Of course, it is not just the size of workspace which matters; it is also the maximum size of arrays. 32-bit APL systems use 32-bit slots to hold array sizes and dimensions. Thus, even if the processor can address more than 32-bits and the APL interpreter provide access to a workspace size greater than 4Gb, it does not follow that the APL interpreter will necessarily handle arrays which will take full advantage of this memory.

For this reason, MicroAPL is introducing a fully 64-bit version of APLX, which will overcome all limitations in workspace and array size for the foreseeable future. APLX64 is this new product. It is initially being made available for Linux and Windows, with a MacOS version following. In APLX64, all array dimensions are 64-bit, so there are effectively no limits other than available memory on the number of elements in any array, or the maximum length along any dimension. Integers are also 64-bit (otherwise you would have to use floating-point numbers to index arrays!). Booleans remain as one bit per element, making it possible to handle huge Boolean arrays without excessive memory requirements. Huge native files and component files are of course fully supported.

We believe this product extends the reach of APL upwards to applications which previously were beyond its reach. These include modelling and simulation using huge data sets, and OLAP applications for analysis and aggregation of large volumes of transactional data. In this new world, you do not need to compromise: you can just load the entire database into your APL workspace (the APLX ⎕SQL system function will be handy here!). In some ways, this is taking APL back to the kinds of application where it used to be strong, but where it lost ground because the amount of data became too big to fit in the workspace. The benefit is direct manipulation, in the APL workspace, of data which other languages have to process piecemeal.

APLX64 is currently in beta, and will be available in the first quarter of 2006.

Kx Systems

Kx Systems has announced new speed gains for its kdb+tick application, achieved by eliminating the latency between data capture and data analysis. Kx, leader in high-performance databases and financial applications, offers kdb+tick worldwide on Linux, Solaris and Windows x64 operating systems.

> "Today's onslaught of tick data streams in 24x7 from multiple exchanges worldwide. The trading firm that acts on this data faster stands to make enterprising trades microseconds ahead of the competition," said Simon Garland, Kx CTO. "Zero latency kdb+tick makes streaming data usable in applications immediately."

Because kdb+tick v. 2.2 removes the traditional delay between the capture of streaming data and its availability for applications, it enables traders to receive every single tick instantly. Applications written using kdb+tick can, for example, update a spreadsheet in real time with every tick that streams in.

By removing the latency between tick data capture and analysis, kdb+tick endows extreme speed applications, such as auto trading and program trading, with more time to execute advanced strategies. It becomes feasible, for example, to perform full depth-of-book program trading on billions of ticks in a single trading day.

Matthew Rock, Director IT at Dresdner Kleinwort Wasserstein, commented "We rely on the capture speed and fast, flawless time-series analysis in kdb+tick to develop unprecedented trading models, to back test trading strategies, and to position DrKW for superior program trading."

Kdb+tick gives traders more time to respond decisively and intelligently to market conditions in real time, and its database heritage takes kdb+tick beyond

ordinary streaming data solutions. Without losing a microsecond of red-hot speed for streaming data processing, kdb+tick also maintains a real-time database that process a million events per second and saves a historical database for business-critical functions such as regulatory audit trails. For more information visit *www.kx.com*.

APLNEWS for Japanese Audience
from Kyosuke Saigusa

We started to distribute APL related news to Japanese audience strictly in Japanese language via our APL programmed Web site (*http://aplcons.com/apl*) in recent months. I would like to introduce some reported items to you, wondering if they are of some interest outside Japan.

An interesting demo program has been completed based on the APL COM interface, which became available with IBM APL2 Windows version released in May this year (Workstation APL2 V2.6).

This program (about 250 lines of APL2 code with only one utility subroutine) will start Excel and allow computation of the data on Excel sheets (regarded as pages of 2 dimension arrays in front of you) interactively from an APL window by simple APL statements handling 2-dimension arrays.

This will take XLS, CSV,ATF files for input and allow parallel processing between APL and Excel because any changes made on Excel side will be reflected in APL processing because the latest data are always read-in prior to APL processing..

The program will run in full licensed interpreter as well as non-licenced APL2 runtime. Any APL interpreter functions in respective environments are usable without restrictions. We intend to use this program to broaden the base of APL users amongst the Excel users.

There are other topics, such as a common utility APL function library based on namespaces which allow even GRAPHPAK as external functions with a lot of merits for APL group users and Windows dialog design scheme for application developers for higher productivity than standard approach.

But APL2-COM demo-program reminds us of the importance of the use of a very small elementary part of APL2 language for general public for practical results. It will be distributed basically free of charge to individuals outside enterprises. Any inquiries are most welcome.

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'):

- 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
ADVOCORP Oy	APL+Win, APL+Link, APL+Linux, APL+Unix, APL*Plus Sharefile	poa	Complete APL2000 and Statgraphics(StatPoint) product range and links to various third party products.
APL Borealis Inc.	Dyalog APL	poa	Distributor of Dyalog APL products from Dyadic
	APL2000	poa	Distributor of APL2000 products
APL Systems IDC SL	APL2000 APL interpreters and tools	poa	Distributor of APL2000 products for the UK Consulting and maintenance for APL applications
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	
Dyalog	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).
IBM APL Products	TryAPL2	free	APL2 for educational or demonstration use. Download from IBM APL2 ftp site or contact APL Products.
	Workstation APL2 V2 Version 2	$1500	AIX, Linux, Solaris, Windows Product 5724-B74
	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	APL2000	poa	Leading distributor of APL2000 products in Denmark
	Dyalog Ltd.	poa	Leading distributor of Dyalog APL products in Denmark
	IBM	poa	Leading distributor of IBM APL & GraphX products in Denmark
	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
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.
	APLX for Linux (commercial)	499	Linux desktop edition of APLX, compatible with Windows and MacOS versions, with full development environment and GUI programming facilities. Runs under RedHat, Debian, Mandrake and SuSe Linux distributions.
	APLX for Linux (personal)	Free	Full version of APLX, can be downloaded free for personal use.
	APL*PLUS	poa	Manugistics
	APL.68000	poa	MicroAPL Ltd
	APL2	poa	IBM
Strand Software	Canada		
	All APL*PLUS products	poa	All APL*PLUS products including upgrades and educational.
	Dyalog and JSoftware products USA	poa	
	Dyalog and JSoftware products	poa	
ubJL GmbH (APL Software Team)	Dyalog APL and Dyalog Systems Ltd. Products	poa	

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Acadvent	ARQUE	poa	Software for the quality assurance of university examinations.
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@email.com.
Assured Systems	FLAIR	poa	Finite loader and interactive rescheduler. Customisable full-function scheduling system. (Available outside Australia by special arrangement only.)
Beautiful Systems	ASF_FILE	$399	Dyalog APL/W auxiliary processor for access to APL*PLUS/PC APL component files (*.ASF).
	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
	SharpPlot for Microsoft .Net	500	*RainPro* rebuilt as a fully-managed .Net DLL for use by APL and non-APL applications alike. May be an attractive alternative for Dyalog 11 and Santa Fe users, as it has much improved image support as well as running rather faster, and not cluttering your workspace with APL sourcecode.
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.
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.
Dynamic Logistics Systems GmbH	MPS	poa	Master Production Scheduling
	FBS	poa	Forecasting and Budgeting System
	DRP	poa	Distribution Requirements Planning
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.

	Olmec	poa	APL GUI environment, providing menu bar, tool bar, status area, navigation sidebar (with treeviews & listviews)and client area. All are configured by simple text files and require no programming. Client area has a "tab wizard" option to provide ordered transaction processing
	Nazca	poa	fast, flexible and reliable static database and editor.
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!		
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
Optima	ServiceLine	poa	A property management system which keeps a record of all outstanding tasks, produces an up to date list of work to-do and Scheduled reminders plus automated standard letters and basic financial control.
	TravelLine	poa	A system designed to control the workload allocation for a fleet of chauffeur vehicles plus a reminder system for fleet management and Local Authority requirements.
	BPA	poa	"Brand Performance Analysis" allows for the modelling of product/brand performance over time and comparison with competitor products.
	DBI	poa	"Database Interrogator" allows for the non technical user to ask sensible questions of a large database such as a questionaire and obtain results tables and graphs quickly, easily and accurately.
Qualedi	Qualedi	$850-$5,500	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
	FAB	free	Training program for the above.
Zark	APL Tutor (PC)	$299	APL computer-based training. Available for APL*PLUS PC & APL*PLUS II. Demo disk $10.
	APL Tutor (MF)	$5000	Mainframe version.
	Zark ACE	$99	APL continuing education. APL tutor news and hotline phone support.
	APL Advanced Techniques....	$59.95	488pp. book, (ISBN 0-9619067-07) including 2-disk set of utility functions (APL*PLUS PC format).
	Communications	$200 pc, $500 mf	Move workspaces or files between APL environments.

APL CONSULTANCY AND DEVELOPMENT

COMPANY	PRODUCT	PRICES(£)	DETAILS
ADVOCORP 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, APL based financial client/server applications, Cognos Planning and ReportNet consultancy.
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.
Assured Systems	Consultancy	poa	APL+Win and/or J and/or VB custom systems development. Conversion of APL+DOS to APL+Win a speciality.
AUSCAN Software	Consultancy and Training	poa	APL software development, training
Ray Cannon	Consultancy	500/day +VAT	APL, C, C++, Assembler, Windows, Graphics on PC and IBM Mainframe. Experience in Insurance, Chemical, and Airline Industries
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.
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
Dyalog	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 Derivatives plc	Consultancy	poa	Financial trading software in Q, K and kdb+
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.
HMW	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work. full members of DSDM consortium and Agile Alliance.
Hoekstra Systems	Consultancy	poa	APL consultancy, programming, etc. Also UNIX system administration
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.

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.
Alastair Kinloch	Consultancy	poa	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
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	poa	Business Forecasting & APL Systems.
Nussbaum gift	Consultancy	poa	IT Consultant with a strong focus on APL.
Optima	Consultancy, Training & Development	poa	We have been in business since 1990 and have a team of APL professionals with many years experience in pharmaceutical, industrial and financial systems on both PC and Mainframe platforms. In addition to pure APL we can also offer assistance with network integration, ACT! customisation and issues concerning the internet and/or web design.
Graeme Robertson	Consultancy & Training	250-500	Design and write custom software. Maintain and upgrade APL systems. Deliver customized APL training courses
Skelton Consulting	Consultancy	poa	K/Q Consultancy
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.
Rex Swain	Consultancy	poa	Independent consultant, 25 years experience. Custom software development, PC and/or mainframe.
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.
		MACfns	MACfns has been released. A suite of over 100 high-accuracy, high-speed assembler code utility functions for APL+Win and APL+DOS, MACfns has been thoughtfully designed and implemented for APL programmers. It includes functions for segmented string manipulation, data restructuring, translation and datatype adjustment, and many numerical computations.Typical speedups range from 4-10 times, with reduced workspace usage,

datatype conservation, identity detection, and greater numerical precision amongst the other benefits. Documentation of MACfns is extremely detailed and thorough.

ubJL GmbH (APL Software Team)	poa	Analyses, design and programming.

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL-385	Typefaces	poa	Variants of the APL2741 typeface available to specification. APL unicode fonts designed and built for all flavours of APL.
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.
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 acam@blueyonder.co.uk.
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.
Right Seat Software	Vox Proxy	$199 (comm)	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.
		$69.95 (edu)	Vox Proxy for educational use.

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 Germany e.V.	Germany	APL Journal	Semi-annual meetings	
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 Material fee 5,000yen a year
Rome/Italy SIG	Roma, Italy			
SE APL Users Grp	Atlanta, Georgia	The APL Planet	Semi Annual meetings	$13 (US), $25 (non-US)
SovAPL	Moscow, Russia	-	Seminars and Annual Meeting	
Toronto SIG	Toronto, Canada		Occasional meetings, APL Skills Database, Toronto Toolkit	

ADDRESSES

ORGANISATION	CONTACT	ADDRESS, TELEPHONE, FAX, EMAIL etc.
Acadvent Ltd.	Dr A M Sykes	171 Gower Road, Swansea SA2 9JH, UK. Tel: 01792-201776 Email: Alan.Sykes@which.net
ACM SigAPL		ACM, 1515 Broadway, 17th Floor, New York, NY 10036, USA (Subs only)
ADVOCORP 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
APL-385	Adrian Smith	Brook House, Gilling East, York YO62 4JJ, UK. Tel: 01439-788385 Email: adrian@apl385.com

APL Bay Area APLBUG	Curtis Jones (Sec)	228 South 15th Street, San Jose, CA 95112-2150, USA Tel: +1 (408) 292-4060 Email: curtis_jones@prodigy.net
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 Consultants Japan	Kyosuke Saigusa	1507 shimomizo, sagamihara, kanagawa, Japan 2290015 Tel:+81-042-778-2127 Fax:+81-042-778-2205
APL Germany e.V.	Dr. Reiner Nussbaum	Buchenerstrasse 78, D-68259 Mannheim, Germany. Email: reiner@nussbaum-gift.de
APL Software\Services	Dick Holt	1570 Heather Field Lane, Earlysville VA 22936 USA. Tel: +1 434-973-4255 email: dick_holt@email.com
APL Solutions Inc	Eric Landau	1107 Dale Drive, Silver Spring, MD 20910-1607 USA Tel: +1 (301) 589-4621 Fax: +1 (301) 589-4618 Email: aplsi@starpower.net
Assured 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
AUSCAN Software Ltd	Richard Procter	PO Box 39, Mansfield, Ontario L0N 1M0 Canada Tel: +1-705-434-1239 Email: rjp@ca.inter.net
Beautiful Systems, Inc.	Jim Goff	PO Box 2235 Jenkintown, PA 19046, USA Tel: +1 (215) 635-0375; Fax: +1 (215) 635-9212 Email: info@beautifulsystems.com
Ray Cannon	Ray Cannon	21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS, UK. Tel: 01252-874697 Email: ray_cannon@compuserve.com
The Carlisle Group	Paul Mansour	544 Jefferson Avenue, Scranton PA, 18510 USA. Tel: +1 570-963-2036
Causeway Graphical Systems Ltd	Adrian Smith	Brook House, Gilling East, YORK, UK. Tel: 01439-788413 Email: adrian@causeway.co.uk
Cinerea AB	Rolf Kornemark	Box 61, S-193 00 Sigtuna, Sweden. Tel/Fax: +46 859 255 421 Email rolf@cinerea.se
ComLog Software	Jeff Pedneau	18728 Bloomfield Road, Olney, MD 20832 USA Tel: +1 (301) 260-1435 Email: jeff@softmed.com
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
Dyalog Ltd.	Morten Kromberg	Grove House, Lutyens Close, Chineham Court, Basingstoke, Hampshire RG24 8AG, UK. Tel: 01256-338461 Fax: 01256-316559 Email: sales@dyalog.com; support@dyalog.com
Dynamic Logistics Systems GmbH	Michael Baas	Wilhelm-Schöffer-Str. 29, 63571 Gelnhausen Germany. Tel: +49 6051 13067 Fax: +49 6051 16142. Email: info@dls-planning.com
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 Derivatives plc	Michael O'Neill	First Derivatives House, Kilmorey Business Park, Kilmorey Street, Newry, Co, Down, N. Ireland BT34 2DH. Tel. 028 3025 2242 Fax 028 3025 2060 Email:enquiries@firstderivatives.com
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 832463 E-mail: martin@gsoft.plus.com
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

I-APL Ltd	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. Email: acam@blueyonder.co.uk
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	11 Bolshaya Monetnaya Str., suite 6, St. Petersburg 197101 Russia. Tel:+7 812 325-9797 Fax: +7 812 233-9685 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
Japan APL Assoc	Juichiro Takeuchi	Keio Univ. Faculty of Sci.&Tech. 3-14-1 Hiyoshi, Kohoku-ku Yokohama Kanagawa, Japan 223-8522. Tel:+81(045)563-1411 Fax:+81(045)566-1617 E-mail:takeuchi@ae.keio.ac.jp
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@kjk-tieto.com
Kx Systems	Simon Garland [Europe]	555 Bryant St., No. 375, Palo Alto CA 94301 USA. Tel: +1 866 kdb fast email: info@kx.com (in Europe simon@kx.com)
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
Alastair Kinloch Ltd	Alastair Kinloch	519 Webster's Land, Edinburgh EH1 2RX, Scotland, UK. Tel: +44 (0)7802 430 202 Email: alastair.kinloch@btinternet.com
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 dbaronet@milinta.com
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants GU32 3PE, UK. Tel: 01730-263843 Email: apl@ellismorgan.co.uk
Dr. Nussbaum gift mbH	Dr Reiner Nussbaum	Buchenerstrasse 78, D-68259 Mannheim, Germany Tel: +49 621 7152190, Email: reiner@nussbaum-gift.de
Optima Systems Ltd	Paul Grosvenor	Optima House, Mill Court, Spindle Way, Crawley, West Sussex, RH10 1TT, UK. Tel: 01293 562700 Fax: 01293 562699 Email:paul@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
Renaissance Data Sys	Ed Shaw	P.O. Box 511, Botsford, CT 06404, USA. Tel: +1 (203) 270-9729 Email: aplbooks@earthlink.com
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
Graeme Robertson	Dr. Graeme D. Robertson	15 Little Basing, Basingstoke, Hants RG24 8AX, UK. Tel: +44 (0) 1256-364071 Email: GraemeDR@nildram.co.uk
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
Skelton Consulting GmbH	Charles Skelton	Weinklinge 9, 70329 Stuttgart, Germany. Email: c.skelton@skelton.de
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
Soliton Associates		Soliton Inc, 44 Victoria Street, Suite 2100, Toronto, ON M5C 1Y2 Tel: +1 (416) 364 9355 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		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
Sykes Systems Inc	Roy Sykes Jr	4649 Willens Ave., Woodland Hills, CA 91364, USA Tel: +1-818-347-5779 Email: roysykes@earthlink.net
Toronto SIG	Richard Procter	P.O.Box 39, Mansfield, Ontario, L0N 1M0, Canada, email: info@torontoapl.ca
ubJL GmbH (APL Software Team)	Kai Jäger	Thusneldastr. 22, 90482 Nürnberg, Germaany. Tel + 49 911 482512; Fax: +49 911 482518 Email jaeger@ubjl.de
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

WORLD WIDE WEB SITES

ACM SigAPL	www.acm.org/sigapl/
ADVOCORP Oy	www.rett.fi/
AFAPL	www.afapl.asso.fr/ (Journal available on line)
APL2000	www.APL2000.com/
APL-385	www.apl385.com
APL Forum	www.aplforum.com
APL Germany e.V.	www.apl-germany.de
APL Journal, Germany	http://www.rhombos.de/shop/a/show/article/?216
APL Borealis Inc.	www.aplborealis.com
APL Consultants Japan	http://aplcons.com
APL to ASCII	see Waterloo Archive
Assured Systems	www.assuredsystems.com.au/
AUSCAN	http://home.ca.inter.net/rjp/auscan/
Eke van Batenburg	wwwbio.LeidenUniv.nl/~Batenburg/index.html
Carlisle Group	http://carlislegroup.com/
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/
Dyalog Ltd	www.dyalog.com/
DynArray	www.dynarray.com/
Dynamic Logistics Systems GmbH	www.dls-planung.de www.dls-planning.com
FinnAPL	www.pyr.fi/apl/
First Derivatives plc	www.firstderivatives.com
Hoekstra Systems	www.HoekstraSystems.ltd.uk/
IBM APL2	www.ibm.com/software/awdtools/apl
Infostroy	www.infostroy.ru
Insight Systems ApS	www.insight.dk/
Japan APL Association	www.ae.keio.ac.jp/lab/soc/takeuchi/japla/
JSoftware Inc	www.jsoftware.com/

KJK-tieto Oy	www.kjk-tieto.com
kx systems	www.kx.com
Lambent Technology	www.lambenttechnology.com
Lescasse Consulting	www.lescasse.com/
Lingo Allegro USA Inc	www.lingo-allegro.com/
MicroAPL Ltd	www.microapl.co.uk/apl
Milinta Inc	www.milinta.com
Nussbaum gift	www.nussbaum-gift.de
Optima Systems Ltd	www.optima-systems.co.uk
Qualedi, Inc.	www.qualedi.com
Renaissance Data	www.aplbooks.com/
Right Seat Software, Inc.	www.voxproxy.com
SigAPL	www.acm.org/sigapl/
Skelton Consulting GmbH	www.skelton.de
Snake Island Research Inc.	www.snakeisland.com
Soliton	www.soliton.com/
Strand Software Inc.	www.strandsoft.com/
Rex Swain	www.rexswain.com/
Toronto SIG (for Toolkit)	www.torontoapl.ca/
ubJL GmbH (APL Software Team)	www.aplteam.de
Waterloo Archive	www.math.uwaterloo.ca/apl_archives/Welcome.html
Jim Weigang	www.chilton.com/~jimw/

DISCOVER

At Play with J: Metlov's Triumph

by Eugene McDonnell (eemcd@mac.com)

A puzzle was recently announced by Frank Buss on the Internet that led to some interesting discoveries. The puzzle is to be found by Googling "Frank Buss Triangle Problem" and then clicking on "Triangles Challenge" or browsing directly to *http://www.frank-buss.de/challenge/* if you prefer. It says:

The challenge is to write a program, which counts all triangles with area >0 in this figure:
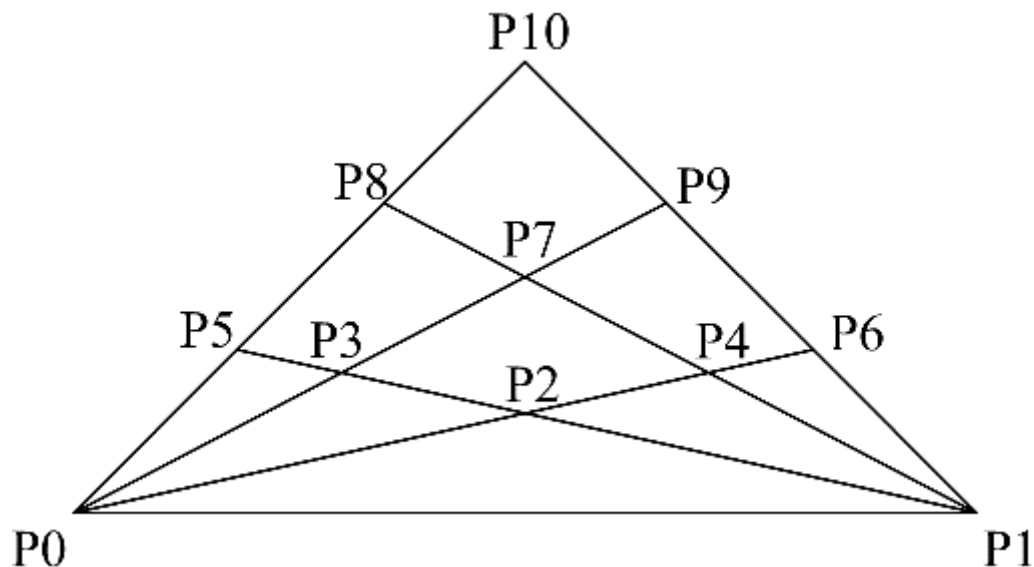
But count only the triangles, which are bounded by lines, like (P0, P7, P8), not all possible connections between the points, like (P7, P8, P9). If anything is unclear, the solution is 27 and looks like this: (see overleaf)

Graphic output is not needed, but you can do what you want. If a GUI or something else is included, it would be nice to write: how long you needed for the pure algorithm and for the rest.

This is not a quantitative, but more a qualitative challenge. Neither the number of lines nor the time (which I can't verify anyway) is important, but I'm interested in good solutions, which show the advantages of the chosen language.

Every program should be documented enough to understand how it works and it should not simply print 27, but somewhere it should read from a file or integrate the points and geometry, so that it is easy to change it for similar problems, for example if another line is added, but it need not to be so general as to count the number of squares.
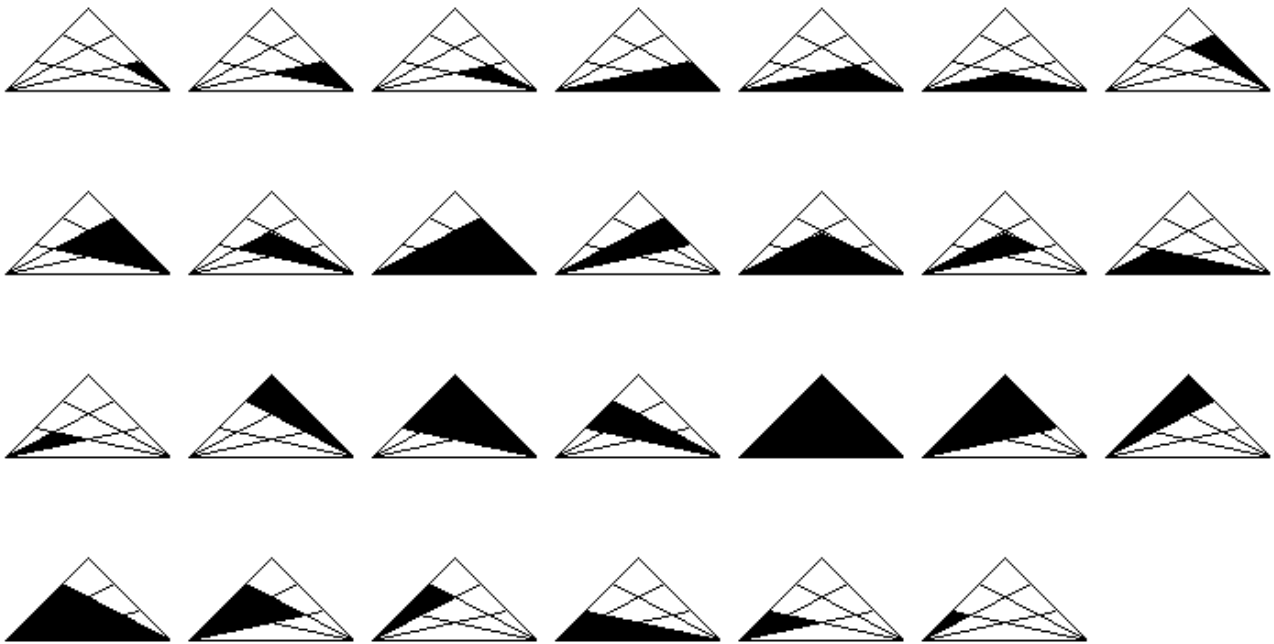
There were 31 entries: The languages they used, the number of entries in that language, and the average number of lines in the programs are tabulated below:

Language	Number of entries	Average number of lines
C++	3	115
Java	4	105
Python	1	94
Haskell	1	93
Ruby	1	75
Scheme	1	66
Awk	1	59
Lisp	17	56
Kogut	1	29
J	1	1

Most of these had a generous amount of documentation along with the actual program. I don't know most of the languages used, but I could come to some conclusions about them. It seems to me that most of the authors were more programmers than mathematicians. Almost all of them tackled the problem as one of establishing the proper way to represent the points, lines, and intersections in the triangle. Most of them gave solutions which were wired in, and their programs could not easily be extended to variations of the problem.

Since Haskell is supposed to be a functional programming language, I thought it might give an interesting and useful result, but I was disappointed. It hard-coded the geometry of the problem, so that it, like many others, couldn't be extended.

The J solution was submitted by Dr. K. L. Metlov. Here it is:

```
  * -: @ * +
```

Metlov is a physicist, with many publications in his field, and he obviously studied the triangle puzzle as a mathematical one. In his notes, one sees that he experiments with variations of the problem, and in a relatively short time had concluded that a simple expression could be formed that would apply to a triangle with any number of lines.

This is a fork, and a dyad, and it is better understood by emphasizing its forkiness.

```
   *     +
    \ /
     |
     *
     |
    -:
```

The arguments are multiplied and added, this product and sum are multiplied, and the product is halved.

I give Metlov's Documentation on the next three pages:

"When both sides of the triangle are divided into an equal number of steps (let's call this number -- n), the number of triangles is n^3 (n to the third power). For the example Frank Buss gives n=3 and the answer is `3^3 = 27`.

When sides are subdivided into a different number of subdivisions, say, n and m, the number of triangles is equal to

$$\tfrac{1}{2}m \times n(m + n)$$

which is integer for any integer m and n.

In J language (see *http://www.jsoftware.com/* for description and download) the first formula is coded and invoked as

```
    nt =: ^&3
    nt 3
  27
```

The second formula is coded and invoked as

```
    nt =: * -: @ * +
    3 nt 3
  27
    2 nt 5
  35
```

The first variant of the program is three characters, the second is 6 characters.

It took me 15-20 minutes of drawing rectangles to derive the formula. J is an array-oriented language, descendent of APL. Therefore, the above programs (without change) can indeed be used to process millions of rectangles very fast. In order to achieve this the arguments must be arrays (of equal length in the second case). For example:

```
    (3 2) nt (3 5)
  27 35
```

How the formula was developed:

Here is the link to the page of notes I made when thinking about the problem. *http://www.livejournal.com/users/dr_klm/51584.html?thread=435072#t435072* and overleaf is a copy of the page.

The direct link is here: *http://galaxy.fzu.cz/~metlov/Triangles_Deriv.gif*

$$N = 2n_1 + 1$$

$$n_1 = m \cdot n_{-1} + (n-1) \cdot m$$

$n = 1$ $m = 1$

$2 \cdot \frac{1}{2} = 1$

$n = 2$ $m = 2$

$$2 \cdot \left(1 + 1 + \frac{1}{2} + \frac{1}{2}\right) + \frac{1}{2} + \frac{1}{2} = 8$$

$n = 3$ $m = 3$

$$N = \left(4 + \frac{1}{2} + \frac{1}{2}\right) + \frac{1}{2} + \frac{1}{2} =$$
$$3 + 1$$

$$2 \times \left[\underbrace{3 + 3 + 3 \cdot \frac{1}{2}}_{\substack{\text{по одному} \\ m \cdot (n-1) + \frac{m}{2}}} + \underbrace{3 + 3 \cdot \frac{1}{2}}_{\substack{\text{по парам} \\ m \cdot (n-2) + \frac{m}{2}}} + \underbrace{3 \cdot \frac{1}{2}}_{\substack{\text{по тройкам} \\ m \cdot (n-3) + \frac{m}{2}}} \right] =$$

$$= \quad 15 \quad + \quad 9 \quad + \quad 3 \quad = 27 \ (!)$$

$$= \sum_{i=1}^{n} \left[m(n-i) + \frac{m}{2} \right] =$$

$$= \frac{m(n-1)}{2} \cdot n + \frac{mn}{2} =$$

$$= \frac{mn^2}{2}$$

$8 \ 5 + 5 + 5$

$\boxed{N_1 = 2 \, \dfrac{n \cdot n^2}{2} = n^3}$ $m \Leftrightarrow n$

$N_2 = \dfrac{mn^2}{2} + \dfrac{nm^2}{2} = \boxed{\dfrac{mn}{2}(m+n)}$

I do not know if that will be enough to communicate the basic idea used for deriving the formula. On the other hand I do not have time to explain it in full detail.

The interesting part occupies the lower left quarter of the page. Triangles are counted separately for two lower corners of the big triangle (left and right) and then the result is multiplied by 2 (if n=m), or added up with exchanging n<->m (if n!=m). To count triangles for one corner I sum up the triangles, occupying all single sub-sectors, the triangles, occupying all pairs of two consecutive sub-sectors, ... three sub-sectors... etc... In this sum, the triangles, which include both left and right corners of the big triangle are counted with weight 1/2 (to note that they will be counted again, in the sum for the other corner).

I ran this procedure for n=3, m=3 approximately in the middle of the page. Then, by induction, wrote a general formula with the sum. The sum is nothing else but an arithmetic progression, which is immediately summed up. Then, with very basic algebra, the final formulas are obtained."

Comment from Frank Buss: *This is a nice solution and the language looks interesting. It is the same concept as the Scheme solution, which uses a formula instead of counting the triangles, but this formula is much easier than the one used in the Scheme solution.*

Functional Programming in Joy and K

by Stevan Apter (sa@nsl.com)

What is Joy?

Joy is a pure, concatenative, functional, scalar programming language.

Joy is pure because it does not contain assignment.

Joy is functional because computation consists of the evaluation of expressions.

Joy is *concatenative* (and not applicative) because

- The elementary well-formed expressions of Joy are monadic functions of a nameless data stack.

- If X and Y are well-formed expressions, then the concatenation of X and Y is well-formed.

- If Z is the concatenation of X and Y, then the value of Z is the composition of the values of X and Y.

For a rigorous exposition of Joy, and for examples of its use, the reader should consult the FAQ, language reference manual, tutorial, and related materials on the official Joy website *http://www.latrobe.edu.au/philosophy/phimvt/joy.html*.

In this note to my interview with Joy's inventor, Manfred von Thun, I describe tcK (*http://www.nsl.com/k/tck/*), a tiny concatenative language modelled on Joy and written in K.

A tiny concatenative K

tcK is a pure, concatenative, functional, *array* programming language.

tcK is a "tiny" version of cK (*http://www.nsl.com/papers/ck.htm*): syntax and display are untranslated K, and the interactive environment is the plain K console.

The *primitives* of tcK are those of K: the twenty dyads

```
~   !   @   #   $   %   ^   &   *   -   _   =   +   |   :   ,   <   .   >   ?
```

and their monadic counterparts

```
~:  !:  @:  #:  $:  %:  ^:  &:  *:  -:  _:  =:  +:  |:  ::  ,:  <:  .:  >:  ?:
```

The *atoms* of tcK are those of K, minus lambdas (defined functions): integers, floats, characters, symbols, null, dictionaries, and lists.

Since tcK is concatenative, everything – primitives, atoms, and lists – is a monadic function of the nameless data stack. For example, the number 12 is a function which takes a data stack and returns it with 12 as the new top element. The "stack diagram" showing the action of the 12 function is:

```
-> 12
```

+ is a monadic function which takes a stack whose top two elements are X and Y and returns it with X and Y replaced by X+Y:

```
X Y -> X+Y
```

Evaluation in tcK uses two stacks, implemented as K lists. The data stack

```
(..;Z)
```

has Z as its top element. The program stack

```
(X;..)
```

has X as its next element.

Since the program stack is a concatenation of monadic functions, it denotes a composition. For example,

```
(2;+;*)
```

is the composition *times of add of 2 of* the data stack. Applied to

```
10 20 30 40 50
```

it returns

```
10 20 30 2080
```

That is,

```
10 20 30 (40 * 50 + 2)
```

Computation consists of evaluating the program stack on the data stack to obtain a new data stack.

Quotations and Combinators

A program in tcK is a list, or in Joy-speak, a *quotation*. Quotations are monadic functions of the stack (everything is), so, applied to the data stack, it returns that stack with itself as the top element.

A *combinator* is a function which expects one or more quotations on the data stack, and applies those quotations in a particular way to the remainder of the stack. Combinators resemble APL operators, or K adverbs.

The simplest combinator is i, which expects a quotation as the top item of the data stack. The action of this combinator is to evaluate the quotation on the remainder of the data stack:

```
    (10;20;30;40;50;(2;+;*);i)
  10 20 30 2080
```

Recursive Combinators

Joy contains several combinators which abstract common patterns of recursion. One such is *linear recursion*, which expects four quotations I, T, E, and F on the data stack. The combinator evaluates the predicate I. If it leaves 'true' on the stack, it evaluates T, else it evaluates E, recurses, and then evaluates F. For example, in Joy the factorial function can be written:

```
    [0 =]  [1 +]  [dup -1 +]  [*] linrec
```

and in tcK:

```
    ((0;=);(1;+);(dup;-1;+);,(*);linrec)
```

Definitions

Joy allows us to create associations between a name and the contents of a quotation:

```
    sqr == dup *
```

The effect of the definition is to add the word 'sqr' to the Joy vocabulary. Note that == is *not* assignment.

The tcK analogue of Joy definition is the function-definition function d. A tcK definition is a *projection* of d onto a quotation:

```
    sqr:d[(dup;*)]
```

The result is a monadic function of the stack which can be used in subsequent evaluations as though it were a primitive of tcK.

An implementation of tcK

The tcK evaluator E is the following dyadic function:

```
E:{*(a .)/(x;y)}
```

E is applied to a data stack x and a program stack y. It calls (a .) repeatedly, initially on (x;y), and thereafter on the result of the previous application, until that result either matches (x;y) or is the same twice in a row.

For convenience, evaluation on the empty data stack is defined as the projection

```
e:E[()]
```

For example,

```
e(10;20;30;+;-)

,-40
```

The application function a is:

```
a:{:[~#y;(x;y);(4:*y)_in 4 7;(f[x;*y];1_ y);(x,1#y;1_ y)]}
```

Again, x is the data stack and y the program stack. If y is empty – all program elements have been processed – then a returns (x;y), which causes E to terminate evaluation and return the final data stack. Otherwise, if the next program element *y is a function or a symbol, f is called to compute the new data stack and *y is dropped from the program stack. Otherwise, the next program element is appended to the data stack and dropped from the program stack.

The function-evaluation function f is:

```
f:{:[(#k)>i:k?y;(v[i]_ x),,y . v[i]#x;y x]}
```

where x is the data stack and y is a single program element to evaluate. k is a list of the forty K primitives, and v is a vector of the corresponding valences, negated for convenience. For example, the dyadic k primitive of equality is k 32, and v 32 is -2.

If y is a K primitive, then the new data stack is constructed by dropping valence-of-y-many elements from the data stack, and appending the result of applying y to those elements.

If y is not a K primitive, then it is either a tcK primitive or a tcK definition.

The tcK primitives are monadic K functions which model the stack operators and combinators of Joy, and the adverbs of K. For example, the Joy operator dup which duplicates the top element of the data stack is written:

```
dup:{x,-1#x}
```

and the K adverb over is written:

```
over:{(-2_ x),,{{*e(y;z),x}[y]/x}.-2#x}
```

The vocabulary of Joy is quite large. Since the purpose of tcK is primarily pedagogical, I've implemented only those operators required by the demonstration problems:

```
dup         X -> X X                   duplicate top of data stack

cons        X [..] -> [X ..]           insert X at head of [..]

swap        X Y -> Y X                 swap top two items of data stack

i           [P] -> P                   evaluate P

linrec      [I] [T] [E] [F] ->         if I then T else: E, recurse, F

right       X Y [F] -> X [F]/:Y        X F/:Y, X F each-right Y

over        X [F] -> [F]/X              F/X, F over X

converge    X [F] -> [F]/X              F/X, R:F X, then R:F R until R ~ X
                                        or R ~ previous R
```

Three Problems

Transitive closure[1]

A K implementation of the classical 'or . and' APL solution:

```
tc:{x|x(|/&)/:x}/
```

tc is the converge of the monadic function

```
{x|x(|/&)/:x}
```

the "noun-verb-adverb" syntax of which is:

```
nvn(vav)an
     ---
      v
```

That is, x is a noun, | and & are transitive verbs, / and /: are adverbs, and the expression (|/&) parses to a transitive verb. In keyword K, tc is:

```
{x or x (or over and) right x} converge
```

We can easily implement tc in prefix form, where expressions involving adverbs are explicit projections of higher-order functions:

```
or:|
and:&
over:{x/y}
converge:{x/y}
right:{y x/:z}
tc:converge[{or[x]right[{over[or]and[x]y}][x]x}]
```

A concatenative language does not have variables. Instead, operators such as dup and swap are used to move items on the data stack into argument position:

```
tc:d[((dup;dup;(&;,(|);over);right;|);converge)]
 e(3 3;0 0 0 1;#;tc)
,(0 0 0
  1 0 0
  1 1 0)
```

Accumulator-Generator[2]

Paul Graham the following problem
(see http://www.paulgraham.com/accgen.html):

> *Write a function foo that takes a number n and returns a function*
> *that takes a number i, and returns n incremented by i.*

We cannot write foo in K, since K lambdas have no state. But tcK programs are lists, and lists have parts which *can* be used to keep state:

```
    acc:d[((+;`acc);cons)]       / accumulator (recursive, must use `acc
                                    instead of acc)
    foo:d[(`acc`i;cons)]         / generator (can use acc and i instead of
                                    `acc and `i)
     e(3;foo)                    / generate a 3-accumulator
,(3;`acc;`i)
     e(3;foo;4;swap;i)           / and accumulate 4
,(7;+;`acc)
     e(3;foo;4;swap;i;5;swap;i) / then accumulate 5
,(12;+;`acc)
```

Quines[3]

A *quine* is a function which prints its own code.

The standard approach is to design a function which indirectly constructs a text-representation of its code. In K (and in many other languages) the ultimate constituents of text are *characters*. But the ultimate constituents of programs are *terms*, so we might expect that a language in which programs are directly available as lists of first-class terms would present the opportunity for a more direct solution.

In tcK, we can define the following quine:

```
    (`dup`cons;`dup;`cons)
```

Evaluation begins by pushing the program `dup`cons on the data stack:

```
    ,`dup`cons
```

Next, dup is evaluated, leaving two items on the stack:

```
    (`dup`cons;`dup`cons)
```

Finally, cons is evaluated, which inserts `dup`cons at the head of the list `dup`cons, leaving

```
    ,(`dup`cons;`dup;`cons)
```

which is the original program.

[1] Adapted from code posted by Greg Heil on the K mailing list.
[2] Joy solution by Martin Young.
[3] Joy solution by Manfred von Thun

tck1.k (http://www.nsl.com/k/tck/tck1.k)

```
// tcK - 1 stack

/ verbs

k:,/+{.:'(x,":";x)}'"~!@#$%^&*_-+=|:<,>.?"  / F1,F2 = (~:;!:;..;~;!;..)
v:-&0 20 20                                 / valences

/ stack operators

dup:{x,-1#x}                                /      X -> X X
cons:{(-2_ x),,{(,x),y}.-2#x}               / X [..] -> [X ..]
swap:{(-2_ x),|-2#x}                        /    X Y -> Y X

/ combinators

i:{E[-1_ x;*-1#x]}                          / [..] -> ..
linrec:{{[x;i;t;e;f]:[E_[x;i];E[x;t];E[_f[E[x;e];i;t;e;f];f]]}[-4_ x].-4#x}
                                            / t if i else: e, recurse, f

/ adverbs (k combinators)

right:{(-3_ x),,{x{*e(y;z),x}[z]/:y}.-3#x}  / X Y f2 -> X f2/:Y
over:{(-2_ x),,{{*e(y;z),x}[y]/x}.-2#x}     /   X f2 ->   f2/X
converge:{(-2_ x),,{{*e(,y),x}[y]/x}.-2#x}  /   X f1 ->   f1/X

/ apply

a:{:[~#y;(x;y);(4:*y)_in 4 7;(f[x;*y];1_ y);(x,1#y;1_ y)]}
                                  / apply if program-stack not empty
f:{:[(#k)>i:k?y;(v[i]_ x),,y . v[i]#x;y x]}  / apply k or tck0 program

/ eval

E:{*(a .)/(x;y)}                            / evaluate y on x
E_:{*-1#E[x;y]}                             / last of evaluate y on x
e:E[()]                                     / evaluate y on ()

/ trace

T:{(a .)\(x;y)}                            / trace evaluation of y on x
t:T[()]                                     / trace evaluation of y on ()

/ define program P:d[(..)] (metalinguistic)

d:{E[y;x]}                                  / evaluate x on y
```

Function Arrays in APL

by Gianluigi Quario (giangiquario@hotmail.com)

The foundations of APL have been be extended to encompass arrays of functions. Nevertheless we have few APL instruments for handling those arrays: this note is an attempt to open up a new panorama to those willing to cultivate this meager field.

Building an Array of Functions

It is possible to build an array-of-functions by means of unnamed namespaces.

You can define a set of unnamed namespaces like this:

```
(ns1 ns2 ns3)←(⎕NS θ)(⎕NS θ)(⎕NS θ)
```

and build a namespace-array

```
nsA←ns1,ns2,ns3
```

nsA is a vector-of-namespaces (shape is 3) and its name class is 2

Let FOO be a function, for example

```
FOO←{3+ω}
```

and make some assignments:

```
ns1.f←FOO ◇ ns2.f←FOO  ◇ ns3.f←FOO
```

Then `nsA.f` is a set of functions . Try

```
    nsA.f
 #. ∇f     #. ∇f     #. ∇f
```

`nsA.f` is a strange object: its name class is 0 ... but

```
    nsA.f 5
 8 8 8
```

and

```
    nsA.⎕NC'f'
 3 3 3
```

Now let us write

```
    FA←nsA.f
```

and obtain

```
    □NC'FA'
3
```

FA is an actual array-of-functions:

```
    FA 5
8 8 8

    FA 5 6 7
8 9 10

    FA 5 6
LENGTH ERROR
```

You can also expunge the involved namespaces and functions; the array-of-functions FA is still alive:

```
    □EX 5 3ρ'ns1ns2ns3nsAFOO'
    FA 5
8 8 8
```

FA has its own actual identity amongst other APL objects.

Array of Functions for the Parallel Headed APLer

The APL scalar functions have a pervasive behaviour over their arguments and are the most "politically correct" functions in a parallel environment. When a function is not scalar, we can force it to behave in a more diligent manner by means of some operator like primitive each ¨ or "saw" or "perv" .

I do not know how you envisage those operators; my basic instinct is to look at them like substitutes for loops.

At the opposite the definition of an array-of-functions carries my mind in a different mood and I actually feel myself thinking in a more holistic way when my attention is forwarded to every kind of arrays.

Consider for example:

```
    ρ¨ (1 2 3)('abcde')
3  5
```

I usually read (i.e. my internal semantics interpreter reads) that statement in this way: "compute the shape of first vector and afterwards of the second one"

But if we afford this task ...

```
nsA←(⎕ns θ) (⎕ns θ)
nsA[1].f←ρ ◇ nsA[2].f←ρ
RHO_parallel←nsA.f
```

... then my mood is much different when I look at:

```
      RHO_parallel (1 2 3)('abcde')
3  5
```

We can avoid the upper manual task of defining many namespaces and afterwards assigning a function by means of a new operator:

```
Parallelized←{((ρω)ρ(⎕NS θ).##).αα ω}
```

and obtain in a more direct manner:

```
      ρParallelized (1 2 3)('abcde') ⎕A ⎕A
3  5  26  26
```

That new operator can be rewritten in a more general way:

```
Parallelized←{
  0∊ρω:ω
  α←{ω}            ⍝ambivalency
  α((ρω)ρ(⎕NS θ).##).αα ω
  }
```

It may be the mate of both monadic and dyadic functions, both primitive and defined. Furthermore it modifies the behaviour of primitive each ¨ operator.

I think that the implementation of each operator by Dyalog APL has some drawbacks.

Let us consider the monadic primitive each:

```
      (0ρ⊂θ)≡ρ¨θ
1
```

```
      (0ρ⊂θ)≡ρ¨''
1
```

```
      (0ρ⊂'')≡ρ¨''
0
```

That I cannot understand! The Dyalog APL language reference says:

> "If the argument Y is empty, the derived function is applied once to the prototype of Y, and the shape of R is the shape of Y.

The thinking behind Dyalog's implementation of each on null arrays is that the prototypical item of the result is determined by the *function*, rather than the argument. Owing to the fact that "each" is an operator, a more consistent behaviour ought to be:

> "If the argument Y is empty, the derived function is the prototype of function operand Lop".

The prototype of any function could be the "TRANSPARENT" function.

You can also consider another example:

```
      {ω[⍋ω]} ¨ θ
RANK ERROR
```

On the contrary it should happen like the following:

```
      {ω[⍋ω]} Parallelized θ
```

gives the zero length numeric vector.

I like to baptize the Parallelized operator with the name "peach", that means parallel each.

```
      peach←Parallelized
```

Building an array of different functions

We are now going to build an array of possibly different functions.

The procedure is similar to what seen beforehand.

A namespace-array is built:

```
      nsA←⎕NS peach 3ρ⊂θ    ⍝length 3 vector
```

and some assignments are made:

```
      nsA[1].f←+ ◇ nsA[2].f←-  ◇ nsA[3].f←÷
      FA←nsA.f
```

Now the array-of-funtions FA can be exploited:

```
      FA 5
5 ¯5 0.2

      FA 3 4 5
3 ¯4 0.2
```

You can see that there is a major duality tie between the array-of-data and the array-of-functions; APL always tries to behave the parallel way:

a) if FA is a (scalar) function and DA is an array-of-data, then FA DA gives an array with the same shape as DA

b) if FA is an array-of-functions and DA is a scalar datum, then FA DA gives an array with the same shape as FA

c) if FA is an array-of-functions and DA is an array-of-data, then FA and DA must have the same shape and FA DA gives an array with the same shape as FA and DA

In Vector Vol.20 No.1 Graeme D. Robertson illustrated a mathematical application of a vector-of-functions related to the velocity of a fluid at a point in 3D space; in traditional notation: V(x,y,z)=(xy, -xy2, yz2)

We define:

```
      fx←{⎕IO←1 ◇ ω[1]×ω[3]} ◇ fy←{⎕IO←1 ◇ -ω[1]×ω[2]*2} ◇
fz←{⎕IO←1 ◇ ω[2]×ω[3]*2}
      nsA←⎕NS peach 3ρ⊂θ    ⍝length 3 namespace-vector

      nsA[1].f←fx ◇ nsA[2].f←fy  ◇ nsA[3].f←fz

      FA←nsA.f
```

and now can exploit the vector-of-functions FA:

```
      FA 1 2 3
3 ¯4 18
```

Tools for handling Arrays of Functions

The Dyalog 10.0 APL interpreter does not allow to handle arrays-of-functions in a direct way.

You cannot use indices:

```
      FA[1]
```

is a pitfall for the interpreter;

```
      1⊃FA
SYNTAX ERROR

      4ρFA
SYNTAX ERROR
```

The structural functions need an effort to be promoted to operator level in order to handle the arrays-of-functions.

But now the arrays-of-functions are a kind of black boxes after they were built.

We may look for some workarounds.

Transformation of a Vector of Functions into a Vector of Namespaces

First of all, let us find a way to obtain an array-of-namespaces from an array-of-functions.

We shall use the FA's canonical representation, which is a class 2 object.

When FA is a vector of functions, its canonical representation is a (complicated) nested array but it is possible extract the built-in functions.

I defined a traditional operator for executing that job; it was not possible to define a function, because its arguments cannot be class 3 objects.

The syntax is

```
    nsArray←(dummyOperand FAtoNS FunctionVector)dummyArg
```

the result is a namespace-array (class 2) where every namespace contains only one function.

The operator FAtoNS looks like complicated because of the complicated structure of canonical representations inside operators. It could be transformed to a Defined function, because the result is a class 2 object; the version here represented allows to show and comment the logical structure.

```
∇
 nsArray←(fdummy FAtoNS
funcArray)dummy;cr;peach;funcName;lasteach;last;true_cr;extract_cr;enlist
   ⍝ transform a single function or a vector of functions into a namespace
array
   ⍝ funcArray is a function(0 rank vector of functions) or a vector of
functions
   ⍝ nsArray is a namespace-Array(vector) where each ns contains 1 function
whose name is f
 cr←⎕CR'funcArray'
 :If 2=⍴⍴cr          ⍝operand is a single function
 :OrIf 2≥|≡cr        ⍝  or a primitive function
     nsArray←{ ⍝returns a namespace with function ¨f¨ embedded
         ns←⎕NS θ ◇ ns.f←�⍵ ◇ ns
     }'funcArray'
 :Else              ⍝operand  is an array of functions
     peach←{ ⍝parallel each operator
         2≠⎕NC'⍺':((⍴⍵)⍴(⎕NS θ).##).⍺⍺ ⍵
         ⍺((⍴⍵)⍴(⎕NS θ).##).⍺⍺ ⍵
     }
     enlist←{⎕ML←0            ⍝ List ⍺-leaves of nested array.
         ⍺←0                  ⍝ default: list 0-leaves.
         ⍺≥¯1+|≡⍵:,⍵          ⍝ all shallow leaves: finished.
         1↓↑,/(⊂⊂⊃⊃⍵),⍺ ∇¨,⍵  ⍝ otherwise: concatenate sublists.
     }
     :If 3∊⍴cr       ⍝ housekeeping:⎕CR contains a namespace reference
     :AndIf '.'≡⊃1↓cr
         cr←⊃¯1↑cr
     :EndIf
     extract_cr←{2>|≡⍵:⍵ ◇ ∇⊃¯1↑⍵}
     true_cr←extract_cr peach cr    ⍝canon.rep of all functions
     nsArray←{ ⍝returns a namespace with function ¨f¨ embedded
         ns funcName←{ ⍝returns fname and ns where function was defined
             ~(,3)≡⍴⍵:θ(⎕FX' ',⍵)
             (⊃⍵)(⎕FX' ',⊃¯1↑⍵)
         }⍵
         0=1↑0⍴funcName:{ns←⎕NS θ ◇ ns.f←�,enlist ⍵ ◇ ns}⍵   ⍝without cr
         ns{                                                ⍝with cr
             0∊⍴⍺:{ns←⎕NS θ ◇ ns.f←�,enlist ⍵ ◇ ns}⍵
             ⍺{ns←⎕NS θ ◇ ns.f←⍺�,enlist ⍵ ◇ ns}⍵
         }funcName
     }peach true_cr
 :EndIf
∇
```

Reshaping an Array of Functions

The following operator allows the return of the shape of an array-of-functions.

The syntax is

```
        shape←(dummyOperand FA_shape FunctionArray)dummyArray
```

the result is a vector (class 2) of the shape of the array-of-functions.

```
∇
 shape←(fdummy FA_shape funcArray)dummy
   ⍝ shape of a fn array: primitive"monadic ρ"is promoted to function level
   ⍝ func is a function(0 rank vector of functions) or a vector of functions
   ⍝ try:
   ⍝    ⎕←('' FA_shape {2×⍵})''
   ⍝    FA2←(2 3 FA_reshape +)''        ⍝FA2 is an array-of-functions
   ⍝    ⎕←('' FA_shape FA2})''
 shape←ρ(+FAtoNS func)θ
 ⍝   ⎕NC'funcArray'←→3     ⎕NC'shape'←→2
∇
```

The following operator allows the reshaping of an array-of-functions.

The syntax is

```
        ArrayFunc←(shape FA_reshape funcArray)dummyArray
```

the result is an array-of-functions (class 3) obtained by reshaping another array-of-functions (class 3).

```
∇
 ArrayFunc←(shape FA_reshape funcArray)dummy;ns
 ⍝reshape an array-of-fns: primitive"dyadic ρ"is promoted to function level
   ⍝ funcArray is a function(0 rank vector of fns) or a vector of fns
   ⍝ ArrayFunc is an Array-of-Functions
     ⍝ try:
     ⍝    FA2←(2 3 FA_reshape +)0       ⍝FA2 is an array-of-functions
     ⍝    FA2←(4 FA_reshape {2×⍵})0     ⍝FA2 is a vector-of-functions
     ⍝    FA3←(1 3 FA_reshape FA2)0     ⍝FA3 is an array-of-functions
 ns←(+FAtoNS funcArray)θ  ⍝the fn_array becomes a namespace_array;⎕NC'ns'←→2
 ArrayFunc←(shapeρns).f
     ⍝   ⎕NC'funcArray'←→3     ⎕NC'ArrayFunc'←→3
∇
```

Indexing an Array of Functions

The following operator allows the return of a sub-Array from an array-of-functions.

The syntax is

```
        ArrayFunc←(indicesArray FA_from funcArray)dummyArray
```

the result is an array-of-functions (class 3) obtained by means of another array-of-functions (class 3).

For the sake of simplicity let us impose that indicesArray is a vector with the same length as the shape of funcArray operand.

```
∇
 ArrayFunc←(indicesArray FA_from funcArray)dummyArray;ns
    ⍝indexing of an array-of-fns: primitive "[ ]"is promoted to fn level
  ⍝ funcArray is a fn(0 rank vector of fns) or a vector of functions
  ⍝ ArrayFunc is an Array-of-Functions
    ⍝ try:
    ⍝    FA2←(2 4 FA_reshape {2×ω})0      ⍝FA2 is a (2×4) array-of-functions
    ⍝    FA3←(1 1 FA_from FA2)0           ⍝FA3 is a (scalar) array-of-fns
    ⍝    FA4←((1 2) (2 3)FA_from FA2)0    ⍝FA4 is a (2×2) array-of-fns
 ns←(+FAtoNS funcArray)θ  ⍝the fn_array becomes a namespace_array;⎕NC'ns'←→2
 :Select ⍴⍴ns
 :Case ,0 ◊ ∘ ⍝length error
 :Case ,1 ◊ ArrayFunc←ns[indicesArray].f
 :Case ,2 ◊ ArrayFunc←ns[⊃1↑indicesArray;⊃¯1↑indicesArray].f
 :Case ,3 ◊ ∘ ⍝ et coetera
 ⍝ et coetera
 :EndSelect
 ⍝ ⎕NC'funcArray'←→3    ⎕NC'ArrayFunc'←→3
 ∇
```

Catenating two Vectors of Functions

By means of the last structural operator we can have arrays-of-functions with different function-items.

The syntax is

```
        ArrayFunc←(LfuncVector FA_catenate RfuncVector)dummyArray
```

The result is a new vector-of-functions (class 3) obtained by means of two vector-of-functions (class 3).

```
∇
 ArrayFunc←(Lfunc FA_catenate Rfunc)dummyArray;Lns;Rns
   ⍝ catenate 2 vectors of functions: primitive "," is promoted to fn level
   ⍝ Lfunc is a fn(0 rank vector of fns) or a vector of fns;same for Rfunc
   ⍝ ArrayFunc is an Array(vector) of Fns whose names are Lfunc and Rfunc
     ⍝ try:
     ⍝    FA←(+ FA_catenate -)0        ⍝FA is a vector of fns : FA 3 ←→ 3 ¯3
     ⍝    FA←({2×ω} FA_catenate -)0    ⍝FA is a vector of fns : FA 3 ←→ 6 ¯3
     ⍝    FB←(× FA_catenate FA)0       ⍝FB is a vector of fns : FB 3 ←→ 1 6 3
 Lns←(''FAtoNS Lfunc)θ  ⍝the left  fn_array becomes a ns_array;⎕NC'Lns'←→2
 Rns←(''FAtoNS Rfunc)θ  ⍝the right fn_array becomes a ns_array;⎕NC'Rns'←→2
 ArrayFunc←(Lns,Rns).f
 ⍝ ⎕NC'Lfunc'←→3    ⎕NC'Rfunc'←→3    ⎕NC'ArrayFunc'←→3
 ∇
```

Some Examples

```
      ⎕←(''FA_shape +)''          ⍝the shape of a primitive function is θ

      ⎕←(''FA_shape {,ω})''       ⍝the shape of a function is θ

      FA←(+ FA_catenate -)0       ⍝FA is a vector of functions : FA 3 ↔ 3 ¯3

      FA←({2×ω} FA_catenate -)0   ⍝FA is a vector of functions : FA 3 ↔ 6 ¯3

      ⎕←(''FA_shape FA)''
2

      FA2←(2 4 FA_reshape FA)0    ⍝FA2 is a (2×4) array-of-functions

      ⎕←(''FA_shape FA2)''
2 4

      FB←(× FA_catenate FA)0      ⍝FB is a vector of functions : FB 3 ↔ 1 6
3

      FB2←(2 6 FA_reshape FB)0    ⍝FA2 is a (2×4) array-of-functions

      FB3←((1 2) (2 3 4) FA_from FB2)0   ⍝FB3 is an array-of-functions
```

References

[1] G.D. Robertson, New Foundations, Vector 20.1(2003) 132-142

[2] Dyalog APL/W version 10.0 Language Reference(2003)

LEARN

A Suduko Solver in J

by Roger Hui

Fill the grid so that each row, column, and 3 by 3 box contains the digits 1 through 9.

```
+-----+-----+-----+
|2 0 0|6 7 0|0 0 0|
|0 0 6|0 0 0|2 0 1|
|4 0 0|0 0 0|8 0 0|
+-----+-----+-----+
|5 0 0|0 0 9|3 0 0|
|0 3 0|0 0 0|0 5 0|
|0 0 2|8 0 0|0 0 7|
+-----+-----+-----+
|0 0 1|0 0 0|0 0 4|
|7 0 8|0 0 0|6 0 0|
|0 0 0|0 5 3|0 0 8|
+-----+-----+-----+
```

Welcome to Sudoku.

Sudoku is a popular puzzle in Japan (*su* is number, *doku* is place), to where it was imported from the U.S. It was popularized in the West by Wayne Gould, a New Zealander living in Hong Kong. He maintains a website (*http://www.sudoku.com*) where you can find descriptions, examples, tutorials, and download a puzzle player. In a November 2004 article in the Times, (*http://www.timesonline.co.uk/*), Gould was quoted as saying that some Sudoku puzzles are so difficult that you can't solve them if your life depended on it.

The following Sudoku solver uses a simple but effective strategy. Even puzzles rated as "very hard" require no more than 15 milliseconds and 30 Kbytes on a 500 MHz Pentium 3 computer.

```
j       =. (]/. i.@#) ,{;~3#i.3
r       =. 9#i.9 9
c       =. 81$|:i.9 9
b       =. (,j{9#i.9) { j
```

```
I        =: ~."1 r,.c,.b
R        =: j,(,|:)i.9 9

regions=: R"_ {"_ 1 ]
free    =: 0&= > (1+i.9)"_ e."1 I&{
ok       =: (27 9$1)"_ -:"2 (0&= +. ~:"1)@regions

ac       =: +/ .*&(1+i.9) * 1: = +/"1

ar       =: 3 : 0
 m=. 1=+/"2 R{y.
 j=. I. +./"1 m
 k=. 1 i."1~ j{m
 i=. ,(k{"_1 |:"2 (j{R){y.) #"1 j{R
 (1+k) i}81$0
)

assign =: (+ (ac >. ar)@free)^:_"1

guessa =: 3 : 0
 if. -. 0 e. y. do. ,:y. return. end.
 b=. free y.
 i=. (i.<./) (+/"1 b){10,}.i.10
 y. +"1 (1+I.i{b)*/i=i.81
)

guess  =: ; @: (<@guessa"1)
sudoku =: guess @: (ok # ]) @: assign ^:_ @ ,

see1    =: (;~9$1 0 0)&(<;.1) @ ({&'.123456789') @ (9 9&$) @ ,
see     =: <@see1"1`see1@.(1:=#@$)
diff    =: * 0&=@}:@(0&,)
```

A grid is the ravel of a `9 9` matrix of cells of `i.10` . A box is a 9-element subset of a grid, the ravel of one of the `3 3` regions.

A region is a row, column, or box. The object of Sudoku is to assign numbers to the zero cells of a grid `x` while leaving unchanged the non-zero cells in `x`, so that each region has exactly the elements `1+i.9` .

`j` are the indices in a ravelled grid for each box. `r` are the indices for each row. `c` are the indices for each column. `b` are the indices for each box. Finally, `I` are the indices in a ravelled grid for regions that contain a cell, for each cell of a grid.

`regions x` computes a `27 9` matrix of the 27 regions of grid `x` . `free x` computes a `81 9` boolean array `y` such that `(<((9*i)+j),k){y` is `1` iff `1+k` can be assigned to

cell `i,j` of grid `x`. `ok` applies to one or more grids and returns a `1` for each valid grid.

`ac` and `ar` apply to the free list of a grid. `ac` assigns numbers to cells that have only one candidate. `ar` looks for a number which occurs exactly once in the candidates for a region, and assigns that to the cell for which it is a candidate. `ac` and `ar` correspond to "forced moves". (When `ac` or `ar` is applied to an "impossible" grid, the result can be assignments that are obviously in error.) `assign` repeatedly applies `ac` and `ar` to one or more grids until there are no more changes.

`guessa x` applies to to grid `x` and returns one or more grids with cells fill in with all possible candidates, for a cell that has the smallest set of candidates. `guess` applies `guessa` to one or more grids and returns all the grids generated thereby.

`sudoku x` finds all solutions for grid `x`. An error is signalled if `x` has no solution.

```
   x=: , 0 ". ] ;._2 (0 : 0)
    2 0 0  6 7 0  0 0 0
    0 0 6  0 0 0  2 0 1
    4 0 0  0 0 0  8 0 0
    5 0 0  0 0 9  3 0 0
    0 3 0  0 0 0  0 5 0
    0 0 2  8 0 0  0 0 7
    0 0 1  0 0 0  0 0 4
    7 0 8  0 0 0  6 0 0
    0 0 0  0 5 3  0 0 8
   )
      see x, sudoku x
   +-------------+-------------+
   |+---+---+---+|+---+---+---+|
   ||2..|67.|...|||283|671|945||
   ||..6|...|2.1|||976|548|231||
   ||4..|...|8..|||415|392|876||
   |+---+---+---+|+---+---+---+|
   ||5..|..9|3..|||567|419|382||
   ||.3.|...|.5.|||834|267|159||
   ||..2|8..|..7|||192|835|467||
   |+---+---+---+|+---+---+---+|
   ||..1|...|..4|||321|786|594||
   ||7.8|...|6..|||758|924|613||
   ||...|.53|..8|||649|153|728||
   |+---+---+---+|+---+---+---+|
   +-------------+-------------+
```

The following phrases show the intermediate steps leading to a solution.

```
     f=: + (ac >. ar)@free            one step of assign
```

```
see t=: f^:a: x
```
forced moves leading from grid x

```
see diff t
```
differences from one grid to the next

```
see assign x
```
same as the last grid above

```
see g=: guess (ok#]) assign x
```
guesses after exhausting forced moves

```
see t0=: f^:a: 0{g
```
forced moves leading from guess 0

```
see diff t0
```
differences from one grid to the next

```
see t1=: f^:a: 1{g
```
forced moves leading from guess 1

```
see diff t1
```
differences from one grid to the next;
note the obviously invalid assignments

Sudoku with Dyalog APL

from John Clark & Ellis Morgan

Introduction
by Adrian Smith

This note summarises two approaches to the Sudoku puzzle which were posted on the Dfns newgroup. Ellis Morgan has a recursive puzzle solver, and John Clark has a backtracking solver and a simple puzzle-generator. The contrast in coding styles is interesting in itself, as is the contrast with the J solution from Roger Hui.

Interestingly, John was also the producer of the marvellous panel discussion film (circa 1974) with Ken Iverson, Adin Falkoff, Larry Breed and others discussing the origins of APL.

John Clark's Generator and Solver

`SOLVE mat` is the call. If you don't have a matrix there are several in the workspace. `EASY`, `MEDIUM`, `HARD`, and `VERY_HARD` were taken from the Times of London. e.g. type `SOLVE MEDIUM`

`PUZZLE nn` will generate a Sudoku matrix with nn zeros to be replaced.

`SOLVE PUZZLE 56` will solve a puzzle with 56 random placed open slots ...


```
BACKTRACKS REQUIRED  44
```

```
    SOURCE                    SOLUTION   CHECKS OUT
0.301 seconds
```

ORDERUP is the main working function. It examines each open slot in a matrix
and returns an n-tuple of (row, col, value(s)) that may be place in that cell. The
CHERRYPICK function will place the value referenced by each 3-tuple. The puzzle
HARD may be solved by just looping CHERRYPICK.

SUDO is the function that either loops CHERRYPICK or goes to a backtracking
system if there are no 3 tuples generated by ORDERUP. Essentially the
backtracking is done by a push pop stack where the state is stored as an array of
arrays in BT.

I first saw APL in 1967 when you could only have 1 character names for functions
and variables. From then to now this is the first application that forced me to write
a backtracking system in APL. Here is the code:

```
      ∇ Z←PUZZLE ZC;A;SOL
[1]    SOL←BUILDONE   ⍝ BUILD RAMDOM FILLED IN PUZZLE
[2]    A←81⍴1 ◊ ZC←ZC?81 ◊ A[ZC]←0   ⍝ VECTOR TO SET ZEROS
[3]    ZC←,SOL        ⍝ RAVEL PERFECT SOLUTION
[4]    Z←9 9⍴A\A/ZC   ⍝ PUT IN THE ZEROS.
      ∇


      ∇ Z←BUILDONE;A;B;C;D;E
[1]    A←3 3⍴B←9?9      ⍝ SET UP A RANDOM BOX
[2]    Z←A,(1⊖A),2⊖A   ⍝ SET UP A ROW
[3]    A←1⌽A           ⍝ GO FOR 2ND ROW
[4]    Z←Z,[⎕IO]A,(1⊖A),2⊖A    ⍝ BUILD 2ND ROW
[5]    A←(3 3⍴B)[;3 1 2]       ⍝ SET FOR 3RD ROW
[6]    Z←Z,[⎕IO]A,(1⊖A),2⊖A    ⍝ BUILD 3RD ROW AND EXIT
      ∇


      ∇ SOLVE MAT;A;B;J;M;B1;B2;B3;B4;B5;B6;B7;B8;B9;BKC;BT;BX;R;Tin
[1]    TIMEIN ◊ SUDO MAT ◊ TIMEOUT
      ∇


      ∇ Z←SUDO MAT
[1]    BKC←0 ◊ BT←θ ◊ SM←MAT  ⍝ INITIALIZE COUNTERS SAVE ARGUMENT
[2]    LP0:CHERRYPICK ⍝ GET THE EASY ONES
[3]    →(0≠⍴A)⍴LP0   ⍝ LOOP BACK FOR MORE EASY ONES
[4]    →(0∊MAT)⍴DMN  ⍝ CHECK IF MORE TO DO
[5]    →0×⍴Z←PCHECK  ⍝ EXIT IF COMPLETE
[6]    DMN:→(3≤⍴B←1⊃J)⍴NXT  ⍝ GO TO BACKTRACKING
[7]     ∘∘∘     ⍝ IF YOU GET HERE IT IS TIME TO CATCH FIRE AND BLOW UP
[8]    NXT:MARKIT       ⍝ MARK STATUS TO START
[9]    LP1:CHERRYPICK   ⍝ WORK ON THE EASY ONES
[10]   ⍕(~0∊MAT)/'→0,⍴Z←PCHECK'   ⍝ FINISH CHECK
[11]   →(3=⍴1⊃J)⍴LP1    ⍝ CHECK FOR MORE EASY ONES
[12]   →(2≥⍴1⊃J)⍴FAP    ⍝ HAVE TO GO FOR BACKTRACKING
[13]   →DMN             ⍝ FORWARD ON NEXT STATE
[14]   FAP:BACKTRACK    ⍝ GO BACK TO LAST WORKING CONDITION
[15]   →LP1
      ∇
```

```
      ∇ CHERRYPICK
[1]     J←ORDERUP MAT   ⍝ GET LIST OF POSSIBLE CHOICES
[2]     →(0=⍴A←(3=+/¨⍴¨J)/J)⍴0    ⍝ EXIT IF NO CHERRIES TO PICK
[3]     SET¨A           ⍝ PICK THE CHERRIES
      ∇


      ∇ Z←ORDERUP M;A;B;C
[1]     Z←⍬    ⍝ LOOK FOR ALL CHOICES FOR EACH CELL
[2]    :For C :In ⍳9      ⍝ CHECK EACH SUB MATRIX
[3]      →(0=⍴A←M ZEROBOX C)⍴NXT  ⍝  EMPTY CELL PRESENT
[4]       Z←Z,(⊂M)PZERO¨A  ⍝ CHOICES FOR EMPTY CELLS
[5]    NXT: :End
[6]     Z←Z[⍋+/¨⍴¨Z]    ⍝ SORT TO PUT CHERRIES UP FRONT
      ∇


      ∇ Z←M ZEROBOX N;A;B
[1]     A←M BOXOUT N      ⍝ PULL SUB BOXES
[2]     Z←(0=,A)/,R∘.,C
[3]     ⍎'B',(⍕N),'←A'
[4]     BX←A
      ∇


      ∇ Z←MAT BOXOUT N;A;B;⎕IO
[1]     ⎕IO←0 ◇ R C←,3 3⊤N-1       ⍝ INITIALIZE
[2]     R←R⊃(1 2 3)(4 5 6)(7 8 9)  ⍝ GET THE ROW SET
[3]     C←C⊃(1 2 3)(4 5 6)(7 8 9)  ⍝ GET THE COLUMN SET
[4]     ⎕IO←1                      ⍝ GO BACK TO THE REAL WORLD
[5]     Z←MAT[R;C]                 ⍝ RETURN THE BOX VALUES
      ∇


      ∇ Z←M PZERO RC;A;B;C
[1]     R C←RC                 ⍝ FIND POSSIBLE VALUES
[2]     A←M[R;] ◇ A←(A≠0)/A     ⍝ ROW VALUES
[3]     B←M[;C] ◇ B←(B≠0)/B     ⍝ COLUMN VALUES
[4]     Z←(A,B),,BX ◇ Z←Z[⍋Z] ◇ Z←1↓(Z≠1⌽Z)/Z    ⍝ COMBINE BOX
[5]     Z←RC,(⍳9)~Z    ⍝ (ROW, COL, POSSIBLE VALUES)
      ∇


      ∇ SET LOC;R;C;X
[1]     R C←2↑LOC    ⍝ GET THE ROW AND COLUMN
[2]     X←+/¯1↑LOC   ⍝ GET THE VALUE TO SET
[3]     →((X∊MAT[R;])∨X∊MAT[;C])⍴0  ⍝ EXIT IF VALUE IS NOT USABLE
[4]     MAT[R;C]←X   ⍝ SET THE VALUE IN THE MATRIX
      ∇


      ∇ Z←PCHECK;A
[1]    ⍝ FOR A PRETTY PRINT OUT OF SOURCE AND SOLUTION
[2]     Z←(FRAME SQZERO ¯1 0↓CKMAT SM)MAB' SOURCE'   ⍝ SET THE SOURCE
[3]     Z←(Z(1⊖8 4↑1 5⍴' => '))                      ⍝ ADD IN A NICE ARROW
[4]     B←¯1 35↑A←CKMAT MAT                          ⍝ CHECK THE SOLUTION
[5]     A←(FRAME ¯1 0↓A)MAB B                        ⍝ FRAME THE SOLUTION
[6]     Z←(Z)(A)                                     ⍝ RETURN FANCY PRINT OUT
[7]     '  BACKTRACKS REQUIRED ',BKC                 ⍝ SHOW BACK COUNT ON TOP
      ∇


      ∇ Z←CKMAT MAT;A;B;C;D;I
[1]     Z←∧/45=+/MAT ◇ Z←Z∧∧/45=+⌿MAT     ⍝ CHECK ROW AND COLUMN SUM
```

```
[2]      :For I :In A←ι9                A CHECK 1..9 IN EACH ROW
[3]        Z←Z^^/A∈MAT[I;]
[4]      :End
[5]      :For I :In A
[6]        Z←Z^^/A∈MAT[;I]              A CHECK 1..9 IN EACH COLUMN
[7]      :End
[8]      I←ORDERUP MAT                   A BUILD THE SUB MATRICES
[9]      D←⍉3 3ρFRAME¨⍉¨B1 B2 B3 B4 B5 B6 B7 B8 B9   A  FRAME EACH BOX
[10]     Z←D MAB⍉1↑Z↓' BAD ' '   SOLUTION   CHECKS OUT'  A ADD COMMENT
     ∇


     ∇ Z←FRAME M;⎕IO
[1]      ⎕IO←1            A ENCLOSE MATRIX IN A FRAME
[2]      M←,Z←⎕AV[231],(⎕AV[226],[1]M,[1]⎕AV[226]),⎕AV[231]
[3]      M[1,(1↓ρZ),(ρM)-(¯1+1↓ρZ),0]←⎕AV[223 222 224 221]
[4]      Z←(ρZ)ρM
     ∇


     ∇ Z←A MAB B
[1]      ⍕(2≠ρρA)/'A←(1,ρA)ρA'
[2]      →((¯1↑ρA)≠¯1↑ρB)ρFX
[3]    OU:Z←A,[⎕IO]B
[4]      →0
[5]    FX:→((¯1↑ρA)>¯1↑ρB)ρWB
[6]      A←((1↑ρA),¯1↑ρB)↑A
[7]      →OU
[8]    WB:→(2=ρρB)ρMT
[9]      B←(1↓ρA)↑B
[10]     →OU
[11]   MT:B←((1↑ρB),1↓ρA)↑B
[12]     →OU
     ∇


     ∇ MARKIT
[1]      SET 3↑B     A CHOOSE FIRST CHOICE
[2]      BT←(⊂(MAT)((2↑B),3↓B)),BT   A PUSH CONDITIONS ON BACK TRACK STACK
[3]      BKC←BKC+1   A INCREMENT BACK TRACKING COUNTER
     ∇


     ∇ SET LOC;R;C;X
[1]      R C←2↑LOC   A GET THE ROW AND COLUMN
[2]      X←+/¯1↑LOC  A GET THE VALUE TO SET
[3]      →((X∈MAT[R;])∨X∈MAT[;C])ρ0  A EXIT IF VALUE IS NOT USABLE
[4]      MAT[R;C]←X  A SET THE VALUE IN THE MATRIX
     ∇


     ∇ BACKTRACK;Y
[1]      →(0≠ρBT)ρBKU
[2]      ∘∘∘∘∘ A YOU ARE DEAD IF YOU GET HERE.
[3]    BKU:MAT←1⊃1⊃BT           A RESET TO PAST CONDITION
[4]      SET 3↑Y←2⊃1⊃BT         A SET THE NEXT VALUE
[5]      (2⊃1⊃BT)←Y←(2↑Y),3↓Y   A REBUILD POSSIBLE CHOICES
[6]      →(3≤ρY)ρ0              A QUIT IF MORE CHOICES LEFT FOR THIS CELL
[7]      BT←1↓BT               A POP THE BACKTRACKING STACK
[8]      BKC←BKC+1             A INCREMENT BACK TRACKING COUNTER
     ∇
```

Ellis Morgan's Solver

`grid ← start 5`   Set up the problem in the London Times of 9 June 2005

`2 show grid`   Check that you have got it right

`pre←result pre_solve grid`   See how hard it could be

`ans ← solve grid`   Solve the problem

`2 show ans`   See the answer

`case9←result pre_solve start 9` Easy probelms are solved by pre_solve

This workspace assumes you can read APL. Look at the comments in "solve", "start", "show", and the other functions to see what is going on.

The Code

```
  grid←{left}start style;index;data
 ⍝ set grid for various starting points per style
 ⍝ style is a valid style number supported by this function

 ⍝ left is needed for style =0, when it is (index data) ...
 ⍝ ... and the values in a ravelled grid are set as grid[index]←data

  grid←,9 9⍴⊂⍳9

 :Select style
 :Case 0 ⍝ user specified
     grid[1⊃left]←2⊃left
 :Case 1 ⍝ medium in paper
     index←1 2 5 8 9,(9+2 8),(18+1 4 6 9),(27+4 6),(36+2 8)
     index,←(45+4 6),(54+1 4 6 9),(63+2 8),72+1 2 5 8 9
     data←5 7 1 4 8 2 6 9 6 2 7 4 9 4 2 1 5 7 3 4 1 3 5 6 1 9 3 4
     grid[index]←data

    .......... lots more examples clipped ..........

 :EndSelect

 grid←,¨grid
 grid←9 9⍴grid


     ∇ text←{style}show grid;row;column;mask
[1]   ⍝ display the grid
[2]   ⍝ style = 0 means as 27 by 27 alpha matrix
[3]   ⍝ style = 1 as a 9 by 9 matrix, showing "known" cells only
[4]   ⍝ style = 2 as a 9 by 9 of known cells, with the squares bordered
[5]   ⍝ style = 3 as a 27 by 27, with cells and squares bordered
[6]
[7]    :If 0=⎕NC'style'
[8]      style←0
[9]    :EndIf
```

```
[10]
[11]
[12]    :Select style
[13]    :CaseList 1 2    ⍝ 9 by 9 , blank if not known
[14]      mask←1=⊃∘ρ¨,grid
[15]      text←9 9ρmask\(⊂'')ρ¨1 0⍕mask/⊃¨,grid
[16]
[17]    :Else ⍝ default or style = 0
[18]      text←27 27ρ' '
[19]      :For row :In ⍳9
[20]        :For column :In ⍳9
[21]          text[(3×row-1)+⍳3;(3×column-1)+⍳3]←display⊃grid[row;column]
[22]        :EndFor
[23]      :EndFor
[24]    :EndSelect
[25]
[26]    text←style showlines text
      ∇


      ∇ text←{style}show grid;row;column;mask
[1]    ⍝ display the grid
[2]    ⍝ style = 0 means as 27 by 27 alpha matrix
[3]    ⍝ style = 1 as a 9 by 9 matrix, showing "known" cells only
[4]    ⍝ style = 2 as a 9 by 9 of known cells, with the squares bordered
[5]    ⍝ style = 3 as a 27 by 27, with cells and squares bordered
[6]
[7]    :If 0=⎕NC'style'
[8]      style←0
[9]    :EndIf
[10]
[11]
[12]    :Select style
[13]    :CaseList 1 2    ⍝ 9 by 9 , blank if not known
[14]      mask←1=⊃∘ρ¨,grid
[15]      text←9 9ρmask\(⊂'')ρ¨1 0⍕mask/⊃¨,grid
[16]
[17]    :Else ⍝ default or style = 0
[18]      text←27 27ρ' '
[19]      :For row :In ⍳9
[20]        :For column :In ⍳9
[21]          text[(3×row-1)+⍳3;(3×column-1)+⍳3]←display⊃grid[row;column]
[22]        :EndFor
[23]      :EndFor
[24]    :EndSelect
[25]
[26]    text←style showlines text
      ∇


      ∇ text←display cell;mask
[1]    ⍝ display the possible cell values as 3 by 3 alphabetic block
[2]    ⍝ eg "display ⊃grid[2;3]" to display the values that you ...
[3]    ⍝ ... can validly put in the third column of the second ...
[4]    ⍝ ... row of the current grid.
[5]
```

```
[6]     cell←,cell
[7]    :If 1=⊃ρcell              ⍝ cell has a single known value
[8]    ⍝   mask←0 1 0 1 1 1 0 1 0  ⍝ as a cross
[9]      mask←9↑¯5↑1              ⍝ in centre of 3 by 3 grid
[10]     text←mask\⊃1 0⍕⊃cell
[11]    :Else                    ⍝ cell has many (or no) possible values
[12]     mask←(⍳9)∊cell
[13]     text←mask\mask/1 0⍕⍳9
[14]    :EndIf
[15]    text←3 3ρtext
     ∇


     ∇ text←style showlines text;lines;mask
[1]   ⍝ horizontal and vertical lines between cells and squares
[2]   ⍝ see comments in "show"
[3]
[4]    →(style∊0 1)↑0
[5]
[6]    lines←'-=▢▢'  ⍝ 2 horizontal (cell,square) and 2 vertical characters
[7]
[8]    :If style=2       ⍝ just the squares in a 9 by 9 grid
[9]      mask←13ρ0 1 1 1
[10]     text←mask⍀mask\text
[11]     mask←(mask=0)/⍳ρmask
[12]     text[;mask]←3⊃lines
[13]     text[mask;]←1⊃lines
[14]    :Else            ⍝ cells and squares in a 27 by 27 display
[15]     mask←37ρ0 1 1 1
[16]     text←mask⍀mask\text
[17]     mask←(mask=0)/⍳ρmask
[18]     text[;mask]←3⊃lines
[19]     text[mask;]←1⊃lines
[20]     mask←((ρmask)ρ3↑1)/mask
[21]     text[;mask]←4⊃lines
[22]     text[mask;]←2⊃lines
[23]    :EndIf
     ∇


     ∇ grid←pre_solve grid;found;old;row;column
[1]   ⍝ "pre-solve" grid, by filling each cell with possible values
[2]   ⍝ only change those cells that have more than one possible value
[3]
[4]   ⍝ repeat until the number of cells whose value is known ...
[5]   ⍝ fails to increase when you apply "valid"
[6]
[7]   ⍝ stop if a cell has no valid values available ...
[8]   ⍝ leaving that cell empty in the returned grid.
[9]
[10]  ⍝ for easy problems "pre_solve" can be the solution ...
[11]  ⍝ ... try "case9←result pre_solve start 9" ...
[12]  ⍝ ... otherwise you will need "solve".
[13]
[14]   found←+/1=⊃∘ρ¨,grid
[15]   old←0
[16]   :While found>old
```

```
[17]      old←found
[18]     :For row :In ι9
[19]       :For column :In ι9
[20]         grid[row;column]←grid valid row column
[21]         :If 0∊⊃∘ρ¨grid
[22]            →0
[23]         :EndIf
[24]       :EndFor
[25]     :EndFor
[26]     found←+/1=⊃∘ρ¨,grid
[27]   :EndWhile
     ∇
```

```
     ∇ grid←result grid;shapes;choice
[1]    ⍝ examine grid to see what we have got
[2]    ⍝ returns the original grid ...
[3]    ⍝... after displaying information about the grid in the session
[4]
[5]    shapes←,⊃∘ρ¨grid
[6]
[7]    :If 81=+/shapes
[8]      choice←'no'
[9]    :Else
[10]      :If 6<+/10⍟shapes
[11]        choice←'10*',(2⍕+/10⍟shapes)~' '
[12]      :Else
[13]        choice←⍕×/shapes
[14]      :EndIf
[15]    :EndIf
[16]
[17]  ⍝ cell wise
[18]   (⍕+/1=shapes),' cells known, ',choice,' cell choices.'
     ∇
```

```
     ∇ answer←solve grid;log
[1]    ⍝ solve a grid where known cells are a single number (vec of len one),
[2]    ⍝ and unknown cells are (⊂ι9)
[3]
[4]    ⍝ it is your business to make sure the grid is validly constructed ...
[5]    ⍝ ... each cell is a non-empty vector of integers ...
[6]    ⍝ ... only the integers 1 to 9 are allowed ...
[7]    ⍝ ... integers can not be repeated within a call.
[8]
[9]    ⍝ it is your business to make sure that the grid represents a valid ...
[10]   ⍝ ... Su Doku problem. The program will stop in "recurse" ...
[11]   ⍝ ... if it thinks there is no valid solution.
[12]
[13]   ⍝ No check is made to see that the solution is unique ...
[14]   ⍝ ... "solve" just returns the first one it finds.
[15]
[16]   ⍝ While "solve" runs stuff will display in your session ...
[17]   ⍝ ... see "recurse" for an explanation.
[18]
[19]    answer log←(,⊂0 0 0)recurse,⊂pre_solve grid
     ∇
```

```
      ∇ result←left recurse grids;grid;shapes;min;row;col;log;answer;which;from
[1]    ⍝ Investigate all possible cell values until a solution emerges.
[2]
[3]    ⍝ "grids" is the grids investigated so far, most recent recursion is last.
[4]
[5]    ⍝ "left"  is the cell values investigated, most recent recursion is last.
[6]    ⍝ ... each entry is a vector of three numbers: row; column; "which" ...
[7]    ⍝ ... where "which" is the index number of the chosen option ...
[8]    ⍝ ... from the options available foer the cell.
[9]
[10]   ⍝ "result" is the grid solution and the cell values that led to it.
[11]
[12]    (⍴left),⊃¯1↑left  ⍝ show the depth of recursion, latest cell setting.
[13]    answer←⊃¯1↑grids  ⍝ current grid
[14]
[15]  next_:
[16]    shapes←,⊃∘⍴¨answer
[17]    result←answer left
[18]   →(∧/1=shapes)↑0      ⍝ if the grid is all known cells, it is a solution
[19]
[20]    :If 0∊shapes        ⍝ have we come down a dead end with no solution?
[21]  further_:
[22]      :If 1<⊃⍴left                    ⍝⍝⍝ is this the first recursion?
[23]        row col which←(⍴left)⊃left
[24]        left←¯1↓left
[25]        grids←¯1↓grids
[26]        answer←(⍴grids)⊃grids         ⍝ step back to the previous grid
[27]        from←⊃answer[row;col]
[28]        :If which<⊃⍴from              ⍝ still some mileage in this path ...
[29]          which+←1                    ⍝ try the next available cell value
[30]             ⍝ which,from             ⍝ display for tough testing
[31]        :Else
[32]          →further_                   ⍝ go back to the previous grid
[33]        :EndIf
[34]      :Else
[35]        ∘∘∘ ⍝⍝⍝ stop here if no solution on the first recursion level
[36]      :EndIf
[37]
[38]    :Else                           ⍝ still on track for a possible solution
[39]      min←shapes⍳⌊/shapes~1          ⍝ first cell with the fewest options
[40]      row col←1+9 9⊤min-1
[41]      which←1                  ⍝ say we are choosing the first option ...
[42]      from←⊃answer[row;col]    ⍝ ... and get the options
[43]    :EndIf
[44]
[45]    answer[row;col]←⊂,which⊃from    ⍝ set the cell to its chosen value ...
[46]   ⍝ ... and call "recurse" again to set another cell
[47]    answer log←(left,⊂row col which)recurse grids,⊂pre_solve answer
[48]    →next_
      ∇
```

APL+WebComponent (Part 2)
Deploy your APL+Win Application on the Web

by Eric Lescasse (eric@lescasse.com)

Introduction

This is the second part of a two-part article about the new APL2000 APL+WebComponent product which allows you to publish your APL+Win applications on the Web. The first part introduced this new technology basics and showed how to setup and port a very small APL+Win application to the Web. In this article we will go further within the technology and port to the Web the following APL+Win application:
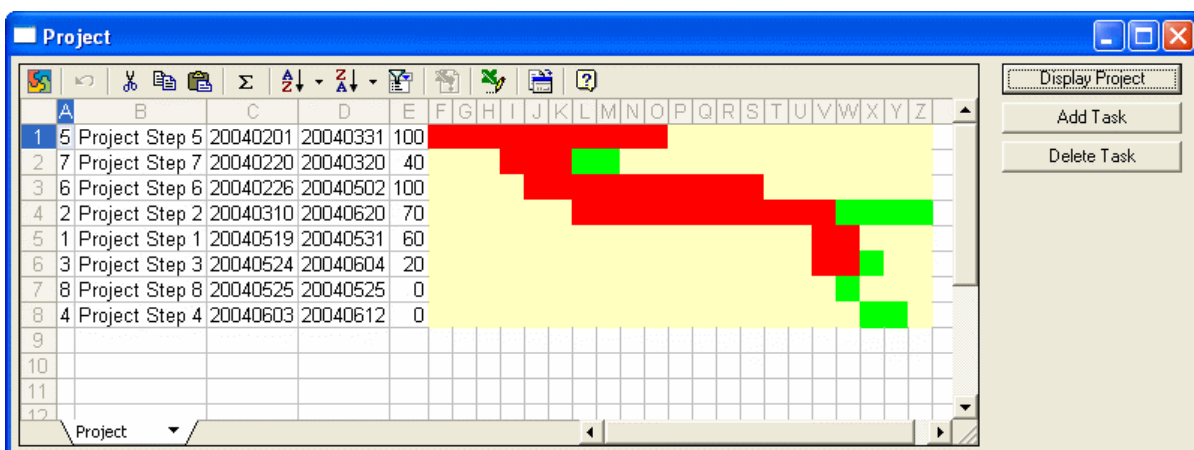
This application displays the various tasks of a Project with a diagram in an OWC (Microsoft Office Web Component) spreadsheet. One can click on the **Add Task** button to add a new task to the Project:

Clicking the **Add** button in the above screen will add the **Project Step 10** task to the project and clicking on the **Display Project** button computes and displays the new Project diagram after sorting the tasks according to their start date:
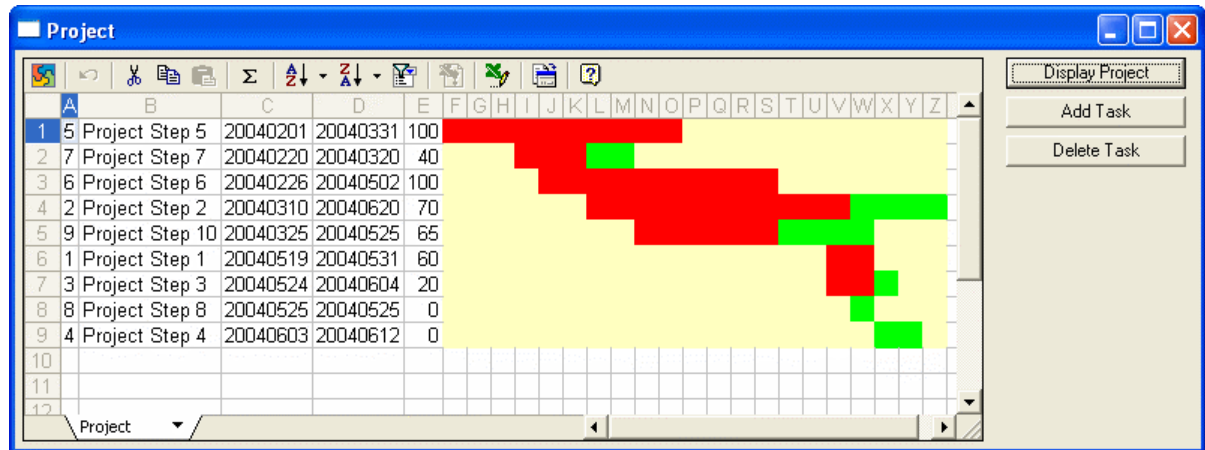
To delete a task, one can click on the **Delete Task** button resulting in the following frame being displayed:
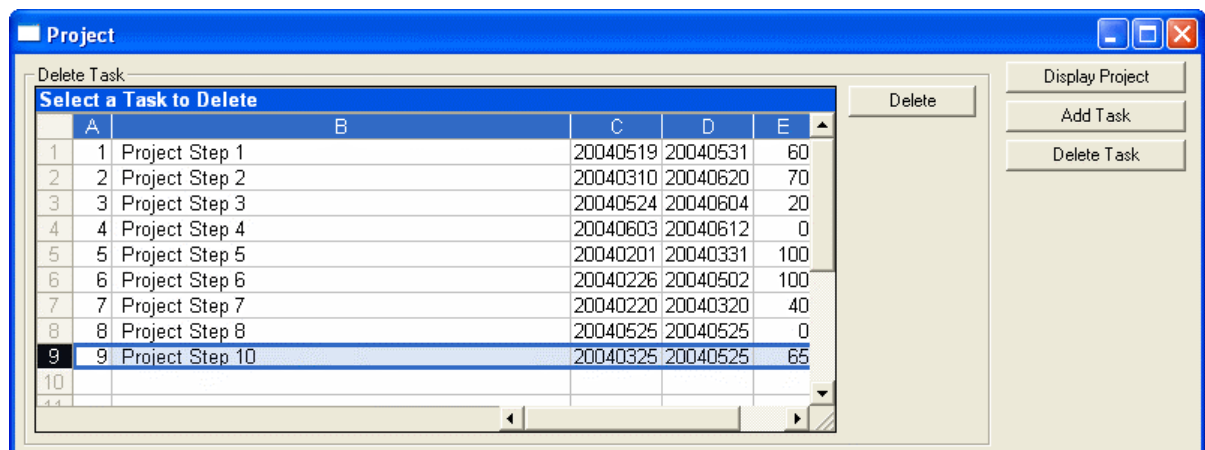
Select a line by clicking on the line number and then click **Delete** to delete the task. Clicking **Display Project** would compute and display the new Project diagram with the task having been deleted.

Downloading and installing the Microsoft OWC Spreadsheet

For this application we will need to use the Microsoft OWC Spreadsheet. This ActiveX object is just a simplified version of Excel especially made by Microsoft for use in the browser, but since it is an ActiveX object, we can also use it within any APL+Win application.

You can download the Microsoft Office Web Components v11 which work with Office 2003 from the Microsoft Web Site. Be sure to select the right language

corresponding to your Office language. To install it, just run the OWC11.EXE file you have downloaded. (Note: if the link is hard to find, start Google and search for: "owc11.exe" and "microsoft".) Note that the Microsoft Office Web Components do not only contain a Spreadsheet object, but also a Charting object and a Database object for publishing graphics or data on the Web.

Finally, if you want to learn how to program the OWC Spreadsheet, there is nothing better than downloading and installing the Microsoft Office Web Components Toolpack.

Preparing the Workspace

This application will be called **owc** so we have created an C:\INETPUB\WWWROOT\LC\WEBSERVICES\OWC.W3 workspace containing all the application functions.

Then we have (p)copied the C:\INETPUB\WWWROOT\LC\WEBSERVICES\APLWS.W3 workspace into OWC.W3 and saved it again. Remember that APLWS.W3 contains the various base functions necessary for APL+WebComponent to work.

Our OWC.W3 workspace is made of 2 functions which we will publish:

- **AutoStart** (this will be the `⎕LX` function)
- **Main** (this will be our Main application function responsible for creating and displaying the interface)

and we will develop several other APL functions which will all run on the Server:

- **TieFile**
- **FileExist**
- **AddTask**
- **Fread**
- **DeleteTask**
- **ComputeProject**
- **DateBase**
- **DATEBASE**
- **DAYOFWK**
- **DATEREP**
- **UniqueFileName**

We will soon explain why we decide to run all these functions on the Server rather than publish them to run in the browser.

Setting up the Web Services with APL+WebServicesController

Rather than going through all the steps (a bit cumbersome) necessary to set up our APL application within the AWS Admin, we will use the new APL2000 **APL+WebServicesController** product to run all the setup steps.

APL+WebServicesController is a COM object and therefore we can use APL+Win to pilot it as required. You can download the Alpha version of APL+WebServicesController from the APL2000 Site (you need to be an APLDN Subscriber, and a Login and Password is required).

Here is the APL+Win Script which sets up the Workspace and the Web Server for our **owc** application:

```
      ∇ CreateOwcServer;Z
[1]    ⍝∇ CreateOwcServer -- Uses the new APL+WebServicesController
[2]    ⍝∇ (c)2004 Eric Lescasse
[3]
[4]    :if 0∊⍴'wsc'⎕wi'self'
[5]        Z←'wsc'⎕wi'Create' 'APL2000.WSC'
[6]    :end
[7]    Z←'wsc'⎕wi'serviceStop'
[8]
[9]    ⍝ Delete Server & Workspace if already exist
[10]   Z←'wsc'⎕wi'DeleteWorkspace' 'owc'
[11]   Z←'wsc'⎕wi'DeleteServer' 'owc'
[12]
[13]   ⍝ Setup new Server
[14]   Z←'wsc'⎕wi'newServer' 'owc'
[15]   Z←'wsc'⎕wi'setServerHost' 'owc' 'localhost'
[16]   Z←'wsc'⎕wi'setServerPort' 'owc' '4000'
[17]   Z←'wsc'⎕wi'setServerPublicHttpDir' 'owc' 'c:\...\webservices'
[18]   Z←'wsc'⎕wi'addServerDefaultFileName' 'owc' 'default.htm'
[19]   Z←'wsc'⎕wi'setEnableDefaultFile' 'owc' 1
[20]
[21]   ⍝ Setup new Workspace
[22]   Z←'wsc'⎕wi'newWorkspace' 'owc'
[23]   Z←'wsc'⎕wi'modifyWorkspaceMaxpool' 'owc' '4'
[24]   Z←'wsc'⎕wi'modifyWorkspaceDebug' 'owc' '1'
[25]   Z←'wsc'⎕wi'modifyWorkspaceLocation' 'owc' 'c:\...\webservices\owc.w3'
[26]
[27]   ⍝ Create and setup new Virtual Directory
[28]   Z←'wsc'⎕wi'newVirtualPath' 'owc' '/jsaveservice/service1.asmx'
[29]   Z←'wsc'⎕wi'modifyServerPathWsid' 'owc' '/jsaveservice/service1.asmx'
            'defaultworkspace' 'owc'
[30]   Z←'wsc'⎕wi'modifyServerPathFunction' 'owc'
            '/jsaveservice/service1.asmx' 'default' 'HTTP_SoapProcess'
[31]   Z←'wsc'⎕wi'addServerPathRargData' 'owc'
            '/jsaveservice/service1.asmx' 'header' 'header'
[32]   Z←'wsc'⎕wi'addServerPathLargData' 'owc'
            '/jsaveservice/service1.asmx' 'data' 'entity-body-utf8'
[33]   Z←'wsc'⎕wi'modifyServerPathResultData' 'owc'
```

```
               '/jsaveservice/service1.asmx' 'r' 'r' 'content-type'
[34]   Z←'wsc'⎕wi'addServerPathResultData' 'owc' '/jsaveservice/service1.asmx'
               'r2' 'soap-envelop-start'
[35]   Z←'wsc'⎕wi'addServerPathResultData' 'owc' '/jsaveservice/service1.asmx'
               'r3' 'soap-body'
[36]   Z←'wsc'⎕wi'addServerPathResultData' 'owc' '/jsaveservice/service1.asmx'
               'r4' 'soap-envelop-end'
[37]
[38]   ⍝ Create and setup new Virtual Directory
[39]   Z←'wsc'⎕wi'newVirtualPath' 'owc' '/owc/xmlfile'
[40]   Z←'wsc'⎕wi'modifyServerPathWsid' 'owc' '/owc/xmlfile'
               'defaultworkspace' 'owc'
[41]   Z←'wsc'⎕wi'modifyServerPathFunction' 'owc' '/owc/xmlfile' 'default'
               'GetXMLFile'
[42]   Z←'wsc'⎕wi'addServerPathRargData' 'owc' '/owc/xmlfile' 'filename'
               'entity-body'
[43]   Z←'wsc'⎕wi'modifyServerPathResultData' 'owc' '/owc/xmlfile' 'r' 'r'
               'document-filename'
[44]   Z←'wsc'⎕wi'addServerPathResultData' 'owc' '/owc/xmlfile' 'r2'
               'document-filename-delete'
[45]
[46]   ⍝ Start Workspace and Server
[47]   Z←'wsc'⎕wi'serviceStart'
[48]   Z←'wsc'⎕wi'startWorkspace' 'owc'
[49]   Z←'wsc'⎕wi'startServer' 'owc'
     ∇
```

After having properly installed the APL+WebServicesController (you just need to run the **APLWSCSetup.msi** installer) run the **CreateOwcServer** function:

```
CreateOwcServer
```

This will result in the following setup added to the AWS Admin console:
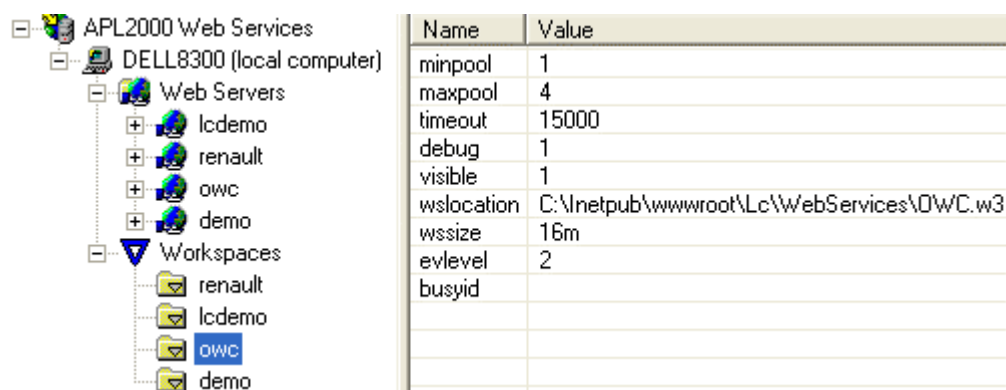
and:
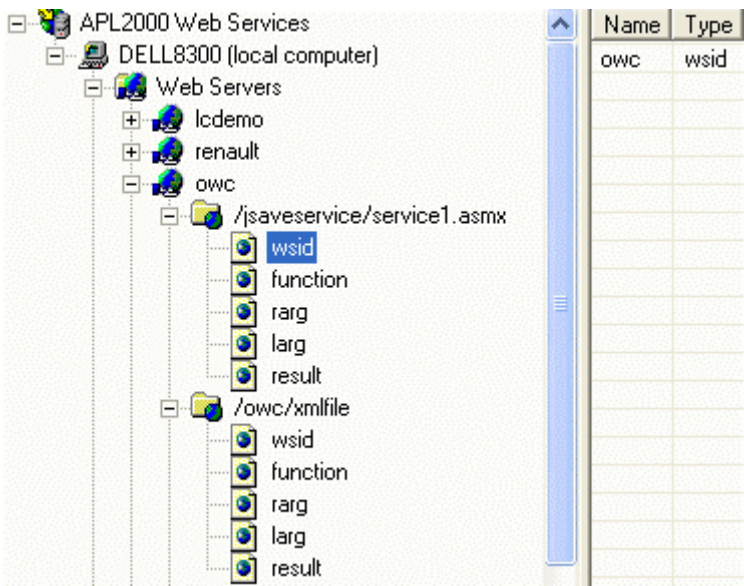
with the following parameters:

Web Server	Virtual Path		Name	Type
owc				
	/jsaveservice/service1.asmx			
		wsid	owc	wsid
		function	HTTP_SoapProcess	function
		rarg	hdr	header
		larg	data	entity-body-utf8
		result	r	content-type
			r2	soap-envelop-start
			r3	soap-body
			r4	soap-envelop-end

	/owc/xmlfile			
		wsid	owc	wsid
		function	GetXMLFile	function
		rarg	filename	entity-body
		larg		
		result	r	document-filename
			r2	document-filename-delete

Publishing the Necessary Functions with JSAVESDK

As done for the Demo application in APL+WebComponent (part 1):

1. Start another APL

2. Load the C:\INETPUB\WWWROOT\LC\WEBSERVICES\JSAVESDK.W3 workspace. The JSAVESDK window pops up

3. Click **Project/Import Workspace** and select your C:\INETPUB\WWWROOT\LC\WEBSERVICES\OWC.W3 application workspace. The functions contained in this workspace get displayed

4. Click on **AutoStart** and then Ctrl+Click on **Main** to select these 2 functions

   Click the **Publish** button
   What you should see so far is:
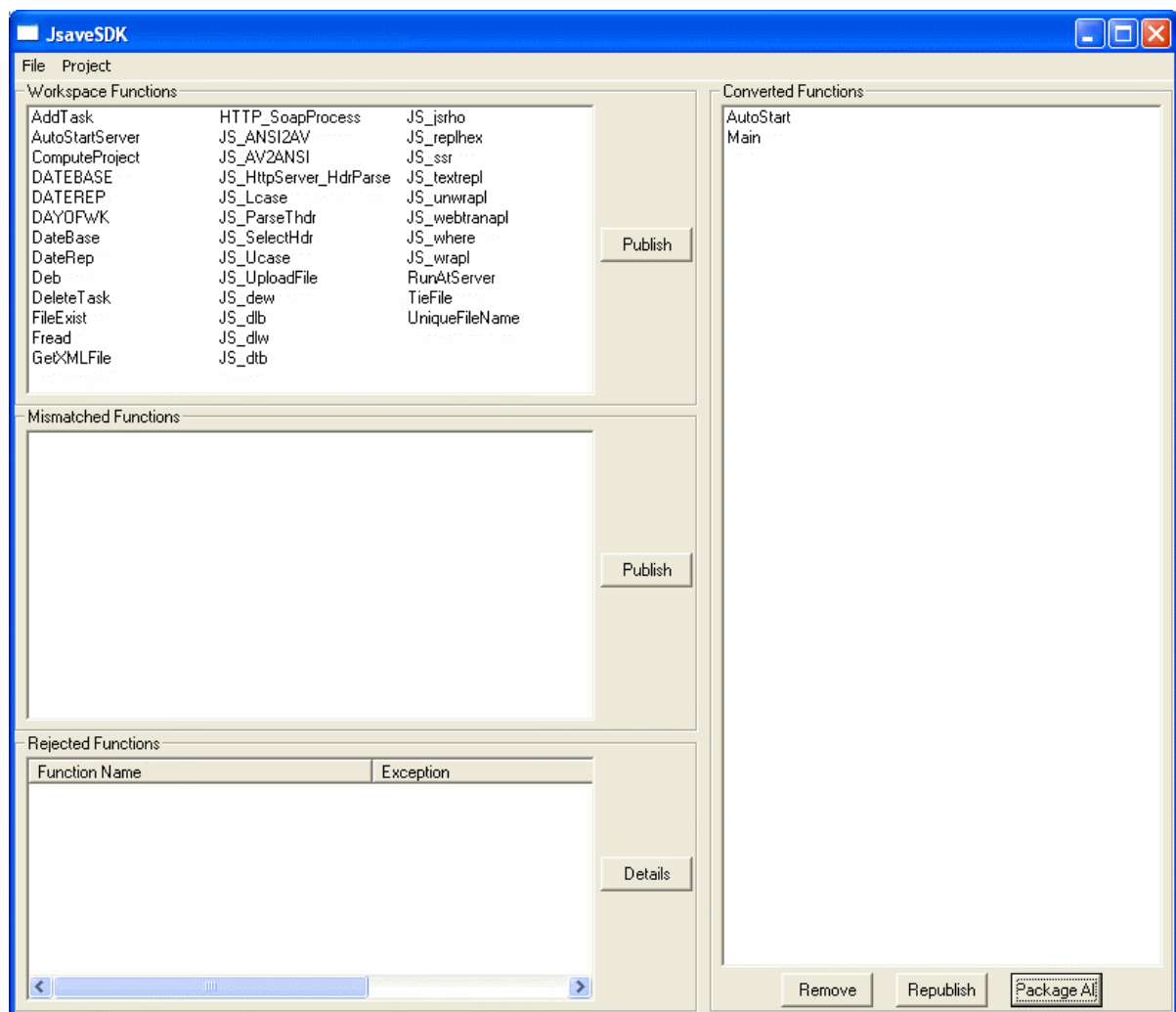
The **Select Dialog Mode** window pops up
Check the **Maintain SI Info** check box in the following dialog then click OK:
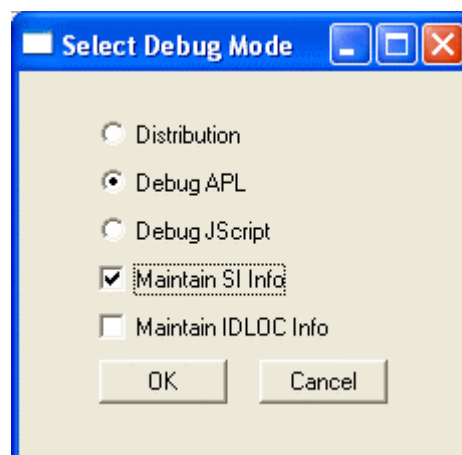
5. Then select **File/Save** and save the JSaveSDK Project as
   C:\INETPUB\WWWROOT\LC\WEBSERVICES\OWC.WJS

Finally click the **Package All** button, fill the next dialog as follows, then click OK:
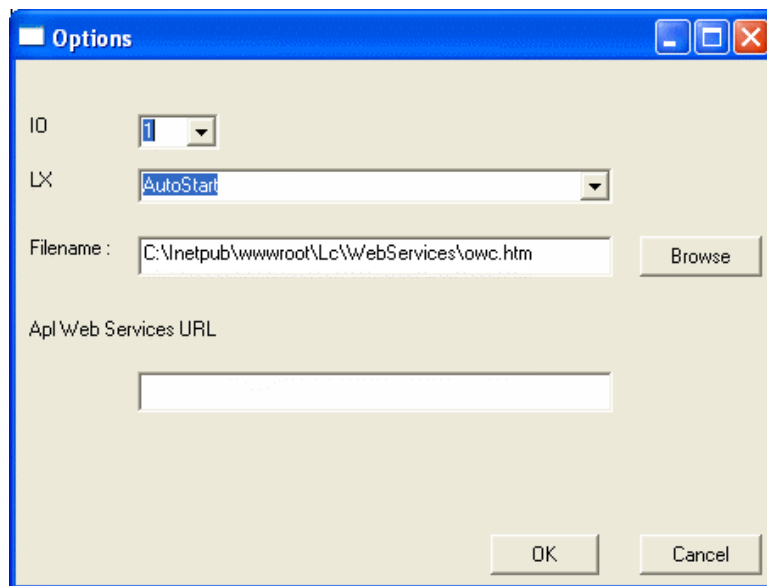
And we are now ready to use the application in the browser!

Using the OWC application in the browser

Start an instance of IE and enter the URL - http://localhost:4000/owc.htm. Here is how it looks:
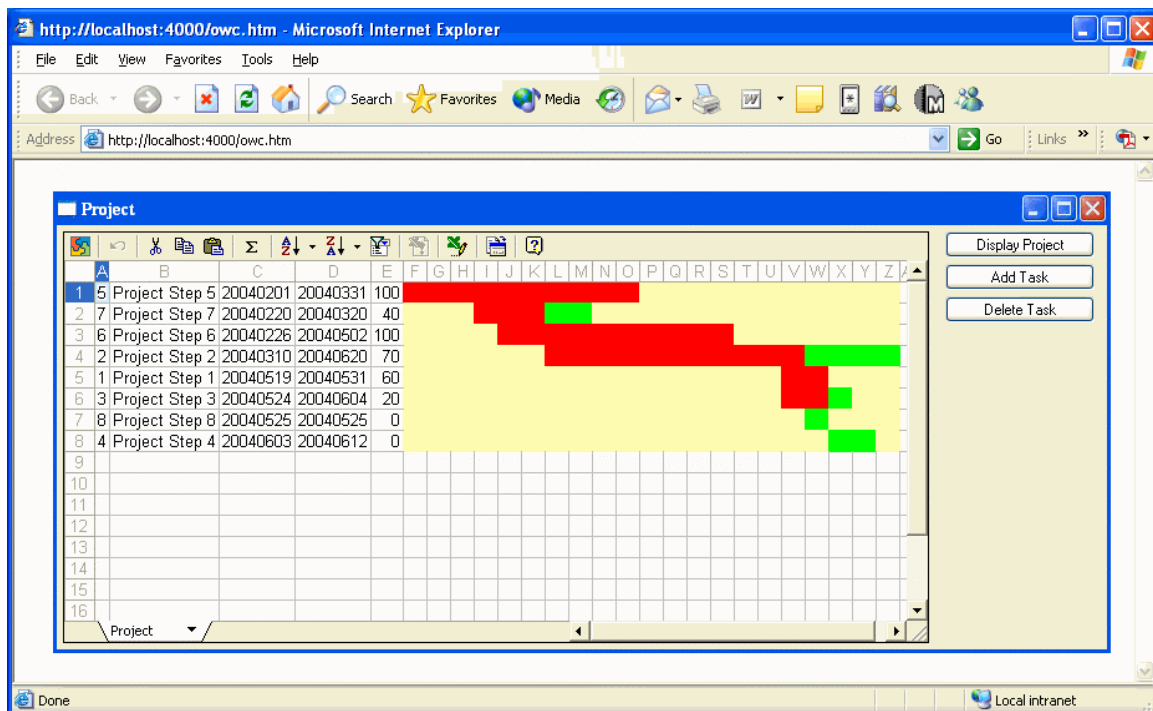
This application is resizable and works exactly the same in the browser as when you start it as a Windows application.

Analysing the Application Code

First let's display the 2 published functions: **AutoStart** and **Main**. Let's analyse the **AutoStart** function first.

```
      ∇ AutoStart;dir;Z
[1]    isΔbrowser←'APL+Js'⎕sysid
[2]    :if isΔbrowser
[3]        Main''
[4]    :else
[5]        AutoStartServer
[6]    :end
      ∇


      ∇ AutoStartServer;dir;Z
[1]    :if 0'#'⎕wi'server'
[2]        Main''
[3]    :else
[4]        ⎕←'I am running on the server!'
[5]        ⍝ Change ⎕chdir from C:\Windows\System32 to the workspace dir
[6]        dir←⌽⎕wsid
[7]        dir←⌽(~^\dir≠'\')/dir
[8]        Z←⎕chdir dir
[9]    :endif
[10]
      ∇
```

We have also displayed the **AutoStartServer** function which is called by **AutoStart**.

Let's comment first about **AutoStart**.

In **AutoStart** we first check ⎕sysid and compare it to 'APL+Js': as a matter of fact ⎕sysid is a way to know if the workspace is run in the browser (⎕sysid is 'APL+Js' in this case) or on the Server (⎕sysid is **APL+Win** in that case). We set a variable isΔbrowser to **1** if we are running in the browser. We will need this information in the **Main** function.

So, if the workspace is run in the browser (i.e. if it is the JScript translated version of the APL code which runs), then we execute Main'' which builds the application interface and displays the Project form in the browser.

On the other hand, if isΔbrowser is 0, this means that we are not running in the browser and this can occur in 2 cases:

- either we have loaded the workspace with the standard APL+Win System in which case '#'⎕wi'server' is 0 and we can start the application (Main '')

- or the workspace has been loaded by an APL+Win ActiveX Server and in this case `'#'⎕wi'server'` returns a non 0 number (a pointer to the APL IUnknown interface) and we know the application is running on the Server: there is no need to display the application there; instead we change the current directory which by default is always C:\Windows\System32 for a COM Server to the application directory ...
  (in our case C:\INETPUB\WWWROOT\LC\WEBSERVICES)

The most important part of this application is of course the **Main** function:

```
      ∇ Main B;Z;height;width;off;M;H;arg;warg;bool;dir;file;colorsmat;tasks;
        spreadcols;lastcol;range;I;J;res;range2;tasks0;ttasks;list;lvwidth;E;F;
        taskid;row;colwidth;rok;data;name;start;end;progress;errmsg
[1]
[2]    :if 1≠arg←B ◇ warg←B ◇ arg←↑B ◇ :end
[3]    :select arg
[4]    :case''
[5]        Z←RunAtServer TieFile 0
[6]        (height width)←'#'⎕wi'*size'
[7]        ⍝ Build the interface
[8]        Z←'fmProject' ⎕wi 'Create' 'Form' ('scale'1)
    ('caption' 'Project')'Hide'
[9]        ⍝ Create an instance of OWC Spreadsheet (OWC11.SpreadSheet)
[10]       Z←'fmProject.owc' ⎕wi 'Create'
    '{0002E551-0000-0000-C000-000000000046}' ('border'1)('DisplayGridlines'0)
[11]       Z←'fmProject.owc' ⎕wi 'Sheets("1").Name' 'Project'
[12]       ⍝ Create buttons
[13]       Z←'fmProject.bnProj' ⎕wi 'Create' 'Button'~
    ('caption' 'Display Project')
[14]       Z←'fmProject.bnProj' ⎕wi 'onClick' 'Main"bnProj.onClick"'
[15]       Z←'fmProject.bnAdd' ⎕wi 'Create' 'Button' ('caption' 'Add Task')
[16]       Z←'fmProject.bnAdd' ⎕wi 'onClick' 'Main"bnAdd.onClick"'
[17]       Z←'fmProject.bnDel' ⎕wi 'Create' 'Button' ('caption' 'Delete Task')
[18]       Z←'fmProject.bnDel' ⎕wi 'onClick' 'Main"bnDel.onClick"'
[19]       ⍝ Create Frame for adding a Task
[20]       Z←'fmProject.frAdd' ⎕wi 'Create' 'Frame' ('scale'1)
    ('caption' 'Add Task') ('visible'0)
[21]       Z←'fmProject.frAdd' ⎕wi 'onResize' 'Main"frAdd.onResize"'
[22]       Z←'fmProject.frAdd.lName' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' 'Task Name')
[23]       Z←'fmProject.frAdd.edName' ⎕wi 'Create' 'Edit' ('scale'1)
[24]       Z←'fmProject.frAdd.lStart' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' 'Start Date')
[25]       Z←'fmProject.frAdd.edStart' ⎕wi 'Create' 'Edit' ('scale'1)
[26]       Z←'fmProject.frAdd.lStart2' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' '(in MM/DD/YY format)')
[27]       Z←'fmProject.frAdd.lEnd' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' 'End Date')
[28]       Z←'fmProject.frAdd.edEnd' ⎕wi 'Create' 'Edit' ('scale'1)
[29]       Z←'fmProject.frAdd.lEnd2' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' '(in MM/DD/YY format)')
[30]       Z←'fmProject.frAdd.lProgress' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' '% Progress')
```

```
[31]        Z←'fmProject.frAdd.cbProgress' ⎕wi 'Create' 'Combo'('style'2 16)
    ('scale'1)('list'(⍕¨5×0,⍳20))
[32]        Z←'fmProject.frAdd.lStatus' ⎕wi 'Create' 'Label' ('scale'1)
    ('caption' '')
[33]        Z←'fmProject.frAdd.bnAdd' ⎕wi 'Create' 'Button' ('scale'1)
    ('caption' 'Add')
[34]        Z←'fmProject.frAdd.bnAdd' ⎕wi 'onClick' 'Main"frAdd.bnAdd.onClick"'
[35]        ⍝ Create Frame for deleting a task
[36]        Z←'fmProject.frDel' ⎕wi 'Create' 'Frame' ('scale'1)
    ('caption' 'Delete Task') ('visible'0)
[37]        Z←'fmProject.frDel' ⎕wi 'onResize' 'Main"frDel.onResize"'
[38]        Z←'fmProject.frDel.owc' ⎕wi 'Create'
    '{0002E551-0000-0000-C000-000000000046}' ('border'1)
    ('DisplayToolbar'0)('DisplayWorkbookTabs'0)
[39]        Z←'fmProject.frDel.owc' ⎕wi 'TitleBar.Visible'1
[40]        Z←'fmProject.frDel.owc' ⎕wi 'TitleBar.Caption'
    'Select a Task to Delete'
[41]        Z←'fmProject.frDel.bnDel' ⎕wi 'Create' 'Button' ('scale'1)
    ('caption' 'Delete')
[42]        Z←'fmProject.frDel.bnDel' ⎕wi 'onClick' 'Main"frDel.bnDel.onClick"'
[43]        ⍝ Make main form the right size
[44]        Z←'fmProject' ⎕wi 'onResize' 'Main"onResize"'
[45]        Z←'fmProject' ⎕wi (⊂'size'),.5 .6×height width
[46]        Z←'fmProject' ⎕wi 'limitwhere' (300÷16) (500÷8)
[47]        ⍝ Display main form
[48]        Main'DisplayProject'
[49]        Z←'fmProject'⎕wi'Show'
[50]
[51]   :case'onResize'
[52]        (height width)←((25×isΔbrowser)0)+16 8×⎕wi'size'
[53]        Z←'fmProject.bnProj'⎕wi(⊂'where'),(5(width-130)21 120)÷16 8 16 8
[54]        Z←'fmProject.bnAdd'⎕wi(⊂'where'),
    ((5+21+5)(width-130)21 120)÷16 8 16 8
[55]        Z←'fmProject.bnDel'⎕wi(⊂'where'),
    ((5+21+5+21+5)(width-130)21 120)÷16 8 16 8
[56]        Z←'fmProject.frAdd'⎕wi(⊂'where'),(7 6 (0⌈height-12)
    (0⌈width-146))÷16 8 16 8
[57]        Z←'fmProject.frDel'⎕wi(⊂'where'),(7 6 (0⌈height-12)
    (0⌈width-146))÷16 8 16 8
[58]        Z←'fmProject.owc'⎕wi(⊂'where'),
    (5 5 (0⌈height-10),(0⌈width-140+10))÷16 8 16 8
[59]
[60]   :case'frAdd.onResize'
[61]        (height width)←('fmProject.frAdd' ⎕wi 'size')×16 8
[62]        off←100
[63]        Z←'fmProject.frAdd.lName' ⎕wi (⊂'where'), ( 17 5 15 off)÷16 8 16 8
[64]        Z←'fmProject.frAdd.edName' ⎕wi
    (⊂'where'), ( 14 (5+off+5) 21 (120⌈width-5+off+5+5+100))÷16 8 16 8
[65]        Z←'fmProject.frAdd.bnAdd' ⎕wi
    (⊂'where'), ( 14 (0⌈width-95) 21 85)÷16 8 16 8
[66]        Z←'fmProject.frAdd.lStart' ⎕wi
    (⊂'where'), ( (17+21+5)5 15 off)÷16 8 16 8
[67]        Z←'fmProject.frAdd.edStart' ⎕wi (⊂'where'),
    ( (14+21+5) (5+off+5) 21 (120⌈180⌊width-5+off+5+5+100))÷16 8 16 8
[68]        Z←'fmProject.frAdd.lStart2' ⎕wi (⊂'where'),
```

```
          ( (17+21+5) (5+off+5+(120⌈180⌊width-5+off+5+5+100)+5) 21 150)÷16 8 16 8
[69]      Z←'fmProject.frAdd.lEnd' ⎕wi (⊂'where'),
          ( (17+21+5+21+5) 5 15 off)÷16 8 16 8
[70]      Z←'fmProject.frAdd.edEnd' ⎕wi (⊂'where'),
          ( (14+21+5+21+5) (5+off+5) 21 (120⌈180⌊width-5+off+5+5+100))÷16 8 16 8
[71]      Z←'fmProject.frAdd.lEnd2' ⎕wi (⊂'where'), ( (17+21+5+21+5)
          (5+off+5+(120⌈180⌊width-5+off+5+5+100)+5) 21 150)÷16 8 16 8
[72]      Z←'fmProject.frAdd.lProgress' ⎕wi (⊂'where'),
          ( (17+21+5+21+5+21+5) 5 15 off)÷16 8 16 8
[73]      Z←'fmProject.frAdd.cbProgress' ⎕wi (⊂'where'),
          ( (14+21+5+21+5+21+5) (5+off+5) 200 60)÷16 8 16 8
[74]      Z←'fmProject.frAdd.lStatus' ⎕wi (⊂'where'),
          ((17+21+5+21+5+21+5+21+5) 5 15 (width-5+5))÷16 8 16 8
[75]
[76]  :case'frDel.onResize'
[77]      (height width)←('fmProject.frDel' ⎕wi 'size')×16 8
[78]      off←100
[79]      Z←'fmProject.frDel.owc' ⎕wi (⊂'where'),
          ( 14 7 (0⌈height-23) (lvwidth←120⌈width-5+5+100))÷16 8 16 8
[80]      Z←'fmProject.frDel.bnDel' ⎕wi (⊂'where'),
          ( 14 (0⌈width-95) 21 85)÷16 8 16 8
[81]      Z←'fmProject.frDel.owc' ⎕wi 'ActiveSheet.Range("a:a").ColumnWidth'3
[82]      Z←'fmProject.frDel.owc' ⎕wi 'ActiveSheet.Range("c:c").ColumnWidth'8
[83]      Z←'fmProject.frDel.owc' ⎕wi 'ActiveSheet.Range("d:d").ColumnWidth'8
[84]      Z←'fmProject.frDel.owc' ⎕wi 'ActiveSheet.Range("e:e").ColumnWidth'5
[85]      colwidth←lvwidth÷8
[86]      colwidth←colwidth-(3+8+8+5)
[87]      colwidth←0⌈colwidth
[88]      colwidth←⌊.5+colwidth
[89]      :if colwidth≠0
[90]          Z←'fmProject.frDel.owc' ⎕wi
          'ActiveSheet.Range("b:b").ColumnWidth' colwidth
[91]          Z←'fmProject.frDel.owc' ⎕wi 'Refresh'
[92]      :end
[93]
[94]  :case'bnAdd.onClick'
[95]      Z←'fmProject.owc' ⎕wi 'visible' 0
[96]      Z←'fmProject.frDel' ⎕wi 'visible' 0
[97]      Z←'fmProject.frAdd' ⎕wi 'visible' 1
[98]
[99]  :case'bnDel.onClick'
[100]     Z←'fmProject.owc' ⎕wi 'visible' 0
[101]     Z←'fmProject.frAdd' ⎕wi 'visible' 0
[102]     Main'FillTasks'
[103]     Z←'fmProject.frDel' ⎕wi 'visible' 1
[104]
[105] :case'bnProj.onClick'
[106]     Z←'fmProject'⎕wi'pointer'11
[107]     Z←'fmProject.frAdd' ⎕wi 'visible' 0
[108]     Z←'fmProject.frDel' ⎕wi 'visible' 0
[109]     Main'DisplayProject'
[110]     Z←'fmProject.owc' ⎕wi 'visible' 1
[111]     Z←'fmProject'⎕wi'pointer'1
[112]
[113] :case'frAdd.bnAdd.onClick'
```

```
[114]        ⍝ Read screen data
[115]        name←'fmProject.frAdd.edName' ⎕wi 'text'
[116]        start←'fmProject.frAdd.edStart' ⎕wi 'text'
[117]        start←(↑¨⎕fi¨(start≠'/')⊂start)[3 1 2]
[118]        start[1]←2000+100|start[1]
[119]        start←100⊥start
[120]        end←'fmProject.frAdd.edEnd' ⎕wi 'text'
[121]        end←(↑¨⎕fi¨(end≠'/')⊂end)[3 1 2]
[122]        end[1]←2000+100|end[1]
[123]        end←100⊥end
[124]        progress←↑⎕fi,'fmProject.frAdd.cbProgress' ⎕wi 'text'
[125]        errmsg←RunAtServer AddTask name start end progress
[126]        :if 0∊⍴errmsg
[127]            'fmProject.frAdd.edName' ⎕wi 'text' ''
[128]            'fmProject.frAdd.edStart' ⎕wi 'text' ''
[129]            'fmProject.frAdd.edEnd' ⎕wi 'text' ''
[130]            'fmProject.frAdd.cbProgress' ⎕wi 'text' ''
[131]            'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#00AA00'
[132]            'fmProject.frAdd.lStatus' ⎕wi 'caption'
                     'Record added to the database!'
[133]        :else
[134]            'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#FF0000'
[135]            'fmProject.frAdd.lStatus' ⎕wi 'caption' errmsg
[136]        :end
[137]
[138] :case'frDel.bnDel.onClick'
[139]        row←'fmProject.frDel.owc' ⎕wi 'ActiveCell.Row'
[140]        tasks←RunAtServer Fread 1 11
[141]        taskid←tasks[row;1]
[142]        Z←RunAtServer DeleteTask taskid
[143]        Main'FillTasks'
[144]
[145] :case'DisplayProject'
[146]        Z←RunAtServer ComputeProject isΔbrowser
[147]        (rok data)←Z
[148]        :if rok = 0
[149]          :if isΔbrowser
[150]                'fmProject.owc' ⎕wi 'xXMLUrl' data
[151]          :else
[152]                'fmProject.owc' ⎕wi 'xXMLData' data
[153]          :endif
[154]        :endif
[155]
[156] :case'FillTasks'
[157]        tasks←RunAtServer Fread 1 11
[158]        :for I :in 1+⍳1↑⍴tasks
[159]          :for J :in ⍳¯1↑⍴tasks
[160]                range←(J⊃'ABCDE'),⍕I
[161]                Z←'fmProject.frDel.owc' ⎕wi
                     ('ActiveSheet.Range("',range,'").Value2') ('')
[162]          :end
[163]        :end
[164]        :for I :in ⍳1↑⍴tasks
[165]          :for J :in ⍳¯1↑⍴tasks
[166]                range←(J⊃'ABCDE'),⍕I
```

```
[167]                Z←'fmProject.frDel.owc' ⎕wi
                     ('ActiveSheet.Range("',range,'").Value2') (⍕tasks[I;J])
[168]          :end
[169]       :end
[170]
[171]  :end
[172]
     ∇
```

Let's explain this function in detail.

First, it is built on a **:select ... :case ...** structure.

When the **Main** argument is an empty character vector, lines **5** to **46** are executed
and the application main form is built.

The interface is made of 3 frames and 3 buttons. Only one of the 3 frames may be
visible at any time: the other 2 are hidden. The 3 buttons help show the
appropriate frame and hide the other 2.

In 2 of the frames an OWC Spreadsheet object is instanciated. Note that in order
for the OWC Spreadsheet object (which is an ActiveX object) to get displayed in
the browser we need to instanciate it using its class id:

```
'fmProject.owc' ⎕wi 'Create' '{0002E551-0000-0000-C000-000000000046}'
```

In order to keep all the code within the **Main** function, we embed all the necessary
handlers within it. For example when a user clicks on the **bnProj** button in the
main form, the following handler is run:

```
Main"bnAdd.onClick"
```

so the **Main** function is called with an argument of **bnAdd.onClick** and the **:select**
control structure branches to line **94** and executes lines **95** to **97**. This technique
makes the code very readable.

Events which we handle here are the **onClick** events on the various buttons, but
also the **onResize** events on the main form and on the Frame objects. This way our
form may be resized by the user (even in the browser) and all controls get nicely
resized or repositioned accordingly.

Another point to notice is that we needed to run a few subroutines to perform
specific tasks like **FillTasks** (to fill the OWC Spreadsheet in the Delete Task frame)
or **DisplayProject** (to compute and display the Project graph).

Rather than making these subroutines, we have made them methods of the **Main** function by encapsulating them as **:case** statements in the **Main** function: this way **Main** contains all the logic necessary to its operation, except for the code sections which need to run on the Server.

The Client Server Decisions and RunAtServer

One of the difficulties you'll bump into when porting APL applications to the Web will be to decide which lines of your code need to run on the Server and which should run on the Client. But first let's explain a little bit more what running on the Client and running on the Server mean.

Every function you have published with JSaveSDK will run on the Client (after having been translated by JSaveSDK to JScript): however your application is installed on a Server and is loaded by APL+Win on the Server when someone starts it in his browser. APL functions which are not published will run on the Server.

In general you are publishing the APL functions which create your application interface and are the main functions of your application: however these functions may call subroutines which you want or need to run on the Server.

The way to do that is to call these functions through the following utility:

```
∇ R←RunAtServer R
∇
```

Here is an example: in our OWC application, we need to use an APL+Win file to store the Project tasks information. Using the file system is not authorized on the Client side for obvious security reasons, therefore we need to open the file on the Server (in any case, most often, a file like the file containing the Project tasks information has to be shared among Web users so it needs to reside on the Server).

To open the file (which we called OWC.SF) we need to write a **TieFile** utility and make it run on the Server.

Here is how we call it:

```
Z←RunAtServer TieFile 0
```

and here is the **TieFile** function which opens (or creates) the **OWC.SF** file:

```
      ∇ R←TieFile dummy;M;H;bool;dir;file;Z;comp2
[1]
[2]    R←0 0ρ''
[3]
[4]    dir←⎕chdir''
[5]    file←dir,'\owc'
[6]
[7]    ⍝ Create or tie the OWC file
[8]    :if FileExist file,'.sf'
[9]        file ⎕fstie 1
[10]   :else
[11]       comp2←'File Structure',⎕tcnl
[12]       comp2←comp2,⎕tcnl,'Comp 1 -- File Description'
[13]       comp2←comp2,⎕tcnl,'Comp 2 -- File Structure'
[14]       comp2←comp2,⎕tcnl,'Comp 3-10 -- (reserved)'
[15]       comp2←comp2,⎕tcnl,'Comp 11 -- Tasks matrix'
[16]       comp2←comp2,⎕tcnl,'            [;1] ↔ Task #'
[17]       comp2←comp2,⎕tcnl,'            [;2] ↔ Task Name'
[18]       comp2←comp2,⎕tcnl,'            [;3] ↔ Task Start Date'
[19]       comp2←comp2,⎕tcnl,'            [;4] ↔ Task End Date'
[20]       comp2←comp2,⎕tcnl,'            [;5] ↔ Task % Progress'
[21]       file ⎕fcreate 1
[22]       Z←'APL+Web Components Project Demo Application File'⎕fappend 1
[23]       Z←comp2 ⎕fappend 1
[24]       Z←(⊂'')⎕fappend¨8ρ1
[25]       Z←(0 5ρ0''0 0 0)⎕fappend 1
[26]
[27]   :end
[28]
      ∇
```

Note that the above function describes the file structure we are using for our very simple OWC.SF application file.

The rules are the following:

- any function called through **RunAtServer** should be monadic and return a result.

  In our case, the **TieFile** function did not need any argument or to return any result, but we still had to add an argument and a result in its syntax; to conform to the above rule.

- The argument to a function called through **RunAtServer** may be a nested vector and its result may also be a nested vector.

Look at the various lines in the **Main** function (displayed above) which call **RunAtServer** to run subroutines on the Server. Line **125** shows an example of passing a nested vector to a function called through **RunAtServer**.

```
[113]        Z←RunAtServer AddTask name start end progress
```

So, the big question is: "when do we need to use **RunAtServer** and when not?"

Here are a few hints to help you make these decisions:

- you must use **RunAtServer** to perform any task which is forbidden on the Client (among these are any file system operations)

- you must use **RunAtServer** whenever your code uses primitives, operators or more generally APL constructs which are not translatable to JScript, though an alternative would be to rewrite these instructions, if possible, using exclusively APL constructs which are supported by JSaveSDK (you can download the precise description of JSaveSDK supported and unsupported APL features from the APL2000 Web Site)

- you should in general use **RunAtServer** to perform anything APL task which is not User Interface related like calculations, database operations, etc.

- you should sometimes use **RunAtServer** when using ActiveX objects because the JSaveSDK support for these ActiveX properties and methods may be limited: in our case, I tried to update and fill the OWC Spreadsheet object to display the project Graph on the Client side, but several properties and methods would not work, so I had to do all this stuff on the Server (see function DisplayProject below)

- finally, you should use **RunAtServer** as often as possible since APL is MUCH faster than JScript, but remember that everything related to your interface should be published, i.e. run on the Client

To better explain this last point, here is an example: we needed to write an **AddTask** function and to run it on the Server to add a new task to our **OWC.SF** since this operation uses a file. It would have been an error to try to read the data input by the Web User within the **AddTask** function. This is easy to understand: the Server does not know about what the Client has done in its browser. Instead we needed to read the data input by the Web User within the **Main** function (which runs on the Client) and to pass these information to the **AddTask** function, hence the following code in the **Main** function:

```
[113] :case'frAdd.bnAdd.onClick'
[114]     ⍝ Read screen data
[115]     name←'fmProject.frAdd.edName' ⎕wi 'text'
[116]     start←'fmProject.frAdd.edStart' ⎕wi 'text'
[117]     start←(↑¨⎕fi¨(start≠'/')⊂start)[3 1 2]
[118]     start[1]←2000+100|start[1]
[119]     start←100⊥start
[120]     end←'fmProject.frAdd.edEnd' ⎕wi 'text'
[121]     end←(↑¨⎕fi¨(end≠'/')⊂end)[3 1 2]
```

```
[122]     end[1]←2000+100|end[1]
[123]     end←100⊥end
[124]     progress←↑⎕fi,'fmProject.frAdd.cbProgress' ⎕wi 'text'
[125]     errmsg←RunAtServer AddTask name start end progress
[126]     :if 0∊ρerrmsg
[127]         'fmProject.frAdd.edName' ⎕wi 'text' ''
[128]         'fmProject.frAdd.edStart' ⎕wi 'text' ''
[129]         'fmProject.frAdd.edEnd' ⎕wi 'text' ''
[130]         'fmProject.frAdd.cbProgress' ⎕wi 'text' ''
[131]         'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#00AA00'
[132]         'fmProject.frAdd.lStatus' ⎕wi 'caption'
                    'Record added to the database!'
[133]     :else
[134]         'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#FF0000'
[135]         'fmProject.frAdd.lStatus' ⎕wi 'caption' errmsg
[136]     :end
```

And here is the **AddTask** function which runs on the Server:

```
    ∇ R←AddTask rarg;name;start;end;progress;tasks;startdate;enddate
[1]    ⍝∇ R←AddTask rarg -- Adds a task to the OWC file
[2]
[3]    R←''
[4]    (name start end progress)←rarg
[5]    (startdate enddate)←⎕split⍉10000 100 100⊤start end
[6]    :if~DATECHECK startdate ◇ R←'Invalid Start Date!  ' ◇  :end
[7]    :if~DATECHECK enddate ◇ R←R,'Invalid End Date!  ' ◇  :end
[8]    :if startdate[1]≠2004 ◇ R←R,'Start year must be 2004!  ' ◇ :end
[9]    :if enddate[1]≠2004 ◇ R←R,'End year must be 2004!  ' ◇ :end
[10]   :if start>end ◇ R←R,'Start date must be before end date!  ' ◇ :end
[11]   :if 0∊ρR
[12]       tasks←⎕fread 1 11
[13]       tasks←tasks⍪(1+⌈/0,tasks[;1])name start end progress
[14]       tasks ⎕freplace 1 11
[15]   :end
    ∇
```

In the **AddTask** function we check the Start date and End date entered on the
client and the **AddTask** function returns an appropriate error message if any of
these dates is not valid for our application. If the dates are valid, we can add the
task to the **OWC.SF** file: this is done on lines **12** to **14**.

In the **frAdd.bnAdd.onClick** event handler, we capture the result of **AddTask** in
the **errmsg** variable and depending on its content we display an error message in
the Add Task frame or inform the user that the record has indeed be added to the
database, in which case we empty the Add Task frame fields.

One interesting point about the **frAdd.bnAdd.onClick** handler is that we have
used a DHTML style property to set the Status label colour:

```
        'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#FF0000'
```

Note that you can use any style, DHTML or JScript property on interface objects as long as you set them as APL User defined properties starting with the 'style_ prefix, followed by the style name (example: 'style_fontFamily , 'style_fontSize , 'style_backgroundColor , etc.). Note that the value you pass to the property should be a valid value for the DHTML or JScript property, hence the '#FF0000' for the colour style here.

Using the OWC Spreadsheet on the Client and on the Server

As I said, not all OWC Spreadsheet properties and methods work on the Client when translated to JScript. However a few of them work fine. In our OWC application, we have used the OWC Spreadsheet both on the Client and on the Server.

Using the OWC Spreadsheet on the Client

Let's first talk about using it on the Client. Look at the **FillTasks** method which role is to fill the OWC Spreadsheet in the Delete Task Frame with our **tasks'** nested array so that the User may select a task to delete:

```
[133]  :case'FillTasks'
[134]      tasks←RunAtServer Fread 1 11
[135]      :for I :in 1+1↑ρtasks
[136]          :for J :in ι¯1↑ρtasks
[137]              range←(J⊃'ABCDE'),⍕I
[138]              Z←'fmProject.frDel.owc' ⎕wi
                      ('ActiveSheet.Range("',range,'").Value2') ('')
[139]          :end
[140]      :end
[141]      :for I :in ι1↑ρtasks
[142]          :for J :in ι¯1↑ρtasks
[143]              range←(J⊃'ABCDE'),⍕I
[144]              Z←'fmProject.frDel.owc' ⎕wi
                      ('ActiveSheet.Range("',range,'").Value2') (⍕tasks[I;J])
[145]          :end
[146]      :end
```

We first need to read the tasks nested array from the OWC.SF file on the Server, which is done through a small trivial **Fread** utility:

```
    ∇ R←Fread A
[1]   R←⍕¨⎕fread A
    ∇
```

(note that we are making each cell a string with the ⍕¨ construct to avoid having to do that on the Client side)

Then we perform 2 double loops on the Client side:

- the first one is to empty the OWC Srpeadsheet

- the second one is to fill it with the tasks nested array

The reasons we have had to do these loops is that the OWC Spreadsheet **Clear** method which was supposed to work on a range of cells did not work when translated through JSaveSDK, and similarly the **Value2** property which normally accepts a nested array to fill a matrix range of cells at once, would not work either when translated through JSaveSDK.

So here is a case where things were not working as expected when translated to JSaveSDK, but where we still could find a workaround to make things work on the Client side. Obviously these loops do not provide us with the best performance possible, especially since JScript is rather slow.

Using the OWC Spreadsheet on the Server

Let's talk now about using the OWC Spreadsheet on the Server side.

You will tell me: what? this is an interface problem and should be running on the Client: how can you make this running on the Server side?

Well, this part is the trickiest in our example, but shows what you can do with APL+WebComponent.

First look at the code which runs in the **Main** function when computing and displaying a new Project graph:

```
[122] :case'DisplayProject'
[123]    Z←RunAtServer ComputeProject isΔbrowser
[124]    (rok data)←Z
[125]    :if rok = 0
[126]       :if isΔbrowser
[127]          'fmProject.owc' ⎕wi 'xXMLUrl' data
[128]       :else
[129]          'fmProject.owc' ⎕wi 'xXMLData' data
[130]       :endif
[131]    :endif
```

Basically almost everything is done on the Server in a **ComputeProject** function (displayed a little further below). We need a function to run on the Server for several reasons here:

- first, this function needs to access the OWC.SF file to read the tasks data

- second, this function needs to perform a bunch of APL calculations to transform the tasks nested array in a Project graph

- third, we wanted to use the OWC Spreadsheet on the Server to make full use of its properties and methods

Here is the **ComputeProject** function:

```
     ∇ R←ComputeProject isΔbrowser;tasks;mindate;maxdate;spandates;boolmat;
       totals;splitmat;daysmat;colorsmat;white;red;green;tasks0;boolmatdone;
       dayofwk;mondays;spreadcols;Z;lastcol;range;range2;I;J;data;filename
[1]    ⍝∇ R←ComputeProject isΔbrowser
[2]    ⍝∇ Comp 11 -- Tasks matrix
[3]    ⍝∇              [;1] ↔ Task #
[4]    ⍝∇              [;2] ↔ Task Name
[5]    ⍝∇              [;3] ↔ Task Start Date
[6]    ⍝∇              [;4] ↔ Task End Date
[7]    ⍝∇              [;5] ↔ Task % Progress
[8]
[9]    tasks←⎕fread 1 11
[10]   tasks0←tasks←tasks[⍋tasks[;3];]              ⍝ sort tasks by Start Date
[11]   tasks[;3 4]←DateBase tasks[;3 4]             ⍝ convert YYYYMMDD dates
[12]   mindate←⌊/tasks[;3]                          ⍝ earliest Start Date
[13]   maxdate←⌈/tasks[;4]                          ⍝ latest End Date
[14]   spandates←mindate+0,⍳maxdate-mindate
[15]   boolmat←(tasks[;3]∘.≤spandates)∧tasks[;4]∘.≥spandates
[16]   boolmatdone←⊃(⊂spandates)∈¨(tasks[;3]-1)+⍳¨⌊.5+.01×tasks[;5]×+/boolmat
[17]   colorsmat←1+boolmat+boolmatdone
[18]
[19]   ⍝ Compute per week
[20]   dayofwk←DAYOFWK DATEREP spandates
[21]   mondays←1,1↓dayofwk=3
[22]   colorsmat←⍉⊃⌈/¨mondays ⎕penclose colorsmat
[23]
[24]   (white red green)←256⊥¨(192 255 255)255(0 255 0)
[25]   colorsmat←(white green red)[colorsmat]
[26]
[27]   spreadcols←256↑,((⊂''),'ABCDEFGHI')∘.,'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[28]
[29]   :if ~0∊⍴'ftmp.owc' ⎕wi 'self'
[30]       Z←'ftmp' ⎕wi 'Delete'
[31]   :endif
[32]   Z←'ftmp' ⎕wi 'Create' 'Form' 'Hide'
[33]   Z←'ftmp.owc' ⎕wi 'Create' '{0002E551-0000-0000-C000-000000000046}'
[34]   Z←'ftmp.owc' ⎕wi 'Sheets("1").Name' 'Project'
[35]
[36]   ⍝ Compute range
[37]   lastcol←(¯1↑⍴tasks)⊃spreadcols
[38]   range←'A1:',lastcol,⍕↑⍴tasks0
[39]   range2←'A:',lastcol
[40]   Z←'ftmp.owc'⎕wi'xActiveSheet.xRange().xValue2'range(⍉tasks0)
[41]   Z←'ftmp.owc'⎕wi'xActiveSheet.xRange().XAutoFit'range2
[42]   ⍝ Install Colors matrix
[43]   range←((1+¯1↑⍴tasks)⊃spreadcols),':',
```

```
              ((¯1↑ρtasks)+¯1↑ρcolorsmat)⊃spreadcols
[44]    range2←(1+(¯1↑ρtasks)+¯1↑ρcolorsmat)⊃spreadcols
[45]    range2←range2,':',range2
[46]    Z←'ftmp.owc'⎕wi'ActiveSheet.Range().ColumnWidth'range 1.5
[47]    Z←'ftmp.owc'⎕wi'ActiveSheet.Range().ColumnWidth'range2 255
[48]    :for I :in ι↑ρcolorsmat
[49]      :for J :in ι¯1↑ρcolorsmat
[50]          range←((J+¯1↑ρtasks)⊃spreadcols),⍕I
[51]          Z←'ftmp.owc'⎕wi'xActiveSheet.xRange().xInterior.xColor'range
              (colorsmat[I;J])
[52]      :end
[53]    :end
[54]
[55]    data←'ftmp.owc' ⎕wi 'xXMLData'
[56]
[57]    :if isΔbrowser
[58]  ⍝     data←'/&/&'TEXTREPL data
[59]        filename←UniqueFileName 'c:\inetpub\wwwroot\lc\webservices\'
[60]        ⎕nuntie ¯1
[61]        filename ⎕xntie ¯1
[62]        0 ⎕nresize ¯1
[63]        filename←(¯3↓filename),'xml'  ⍝ the extn is .tmp, change to .xml
[64]        filename ⎕xnrename ¯1
[65]        data ⎕nappend ¯1
[66]        ⎕nuntie ¯1
[67]        filename←(ρ'c:\inetpub\wwwroot\lc\webservices\')↓filename
[68]  ⍝     R←0 ('http://www.lescasse.com:9000/owc/xmlfile?',filename)
[69]        R←0 ('http://localhost:4000/owc/xmlfile?',filename)
[70]    :else
[71]  ⍝     data←'/&/&'TEXTREPL data
[72]        R←0 data
[73]    :end
[74]
[75]    'ftmp'⎕wi'Delete'
[76]
[77]
     ∇
```

The **ComputeProject** function contains 3 parts:

- lines **9** to **25,** the computation part, takes the **tasks** nested array and converts it to a colours matrix for the Project graph: note that we are using subroutines like **DateBase**, **DATEBASE**, **DATEREP** and **DAYOFWK**, the latter 3 ones coming from the **DATES** workspace delivered with APL+Win. Since these functions are called by a function running on the Server, they automatically also run on the Server and we don't need to use **RunAtServer** again to call them.

  On line **11** the YYYYMMDD dates in the **tasks** nested array are converted to number of days since January 1 1900. The **spandates** variable is computed on line **14** and contains all the dates from the start of the first task to the end of the last task. The **boolmat** variable contains one line per task, one column per **spandates** and a 1 for each day between the task start date and end date. The

boolmatdone variable has the same dimensions as **boolmat** and is the same as **boolmat** except that it contains 1s only for the dates corresponding to the task % which is done.

Finally on line **20** to **22**, we reduce the colours matrix to one cell per task and per week, instead of one cell per task and per day, in order to reduce the number of columns we will use in the OWC spreadsheet.

- lines **21** to **53** are used to create a new instance of the OWC Spreadsheet object on the Server in an invisible form and to fill it with the **tasks** data and with the Graph, i.e. the colours matrix we have just computed: we do this on the Server exactly as we would have liked to do it on the Client.

Now you have to understand that this OWC Server Spreadsheet has nothing to do with the one displayed on the Client in the browser, but it is precisely the same kind of object.

Once the OWC Server Spreadsheet is populated with data, we get its complete content, including all its formatting, in an APL variable called **data** by invoking its **xXMLData** property on line **55**. The **data** variable now contains an XML representation of our OWC Server Spreadsheet content.

- lines **57** to **73** are used to transfer this XML content back to the Client.

One tricky aspect here is that we have to distinguish the 2 following cases: we may be running the application from the browser, or we may be running in a raw APL session just for tests purposes. The tricky thing is that we CANNOT use `⎕sysid` within the **ComputeProject** function to determine if we are running from the browser or from a raw APL session. Guess why? This is because in both cases `⎕sysid` will return **'APL+Win'**. The reason is that when you run a function on the Server through **RunAtServer**, you are running it in a standard APL+Win session on the Server. How do we solve this problem? The trick is simple: since `⎕sysid` returns the right information when we are running on the Client, we just need to pass its Client value to the **ComputeProject** function as its argument. More precisely we are passing here the `isΔbrowser` variable which reflects the `⎕sysid` value, as an argument to the **ComputeProject** function before calling it through **RunAtServer**.

When we are running on the Server, we need to create a native file and to populate it with the **data** variable: this is done through lines **55** to **66**. And we return a return code of 0 and the following string to the client:

```
'http://localhost:9000/owc/xmlfile?',filename
```

where **filename** is the name of the XML file we just created.

Remember that we have created a virtual path called **/owc/xmlfile** as follows in the APL Web Services Configuration Console:

Web Server	Virtual Path		Name	Type
owc				
	/owc/xmlfile			
		wsid	owc	wsid
		function	GetXMLFile	function
		rarg	filename	entity-body
		larg		
		result	r	document-filename
			r2	document-filename-delete

The role of this **/owc/xmlfile** Virtual Path is to send the right XML file from the Server to the Client. The **GetXMLFile** function is very simple and just returns the complete name of the file:

```
      ∇ r←GetXMLFile filename
[1]
[2]    r←('c:\inetpub\wwwroot\lc\webservices\',filename) 1
[3]
      ∇
```

So let's look at the **DisplayProject** method in the **Main** function:

```
[122] :case'DisplayProject'
[123]     Z←RunAtServer ComputeProject is∆browser
[124]     (rok data)←Z
[125]     :if rok = 0
[126]        :if is∆browser
[127]            'fmProject.owc' ⎕wi 'xXMLUrl' data
[128]         :else
[129]            'fmProject.owc' ⎕wi 'xXMLData' data
[130]         :endif
[131]     :endif
```

It runs **ComputeProject** on the Server with an argument of `isΔbrowser` (which is 1). The **ComputeProject** function returns the following string to the client **data** variable:

```
'http://localhost:4000/owc/xmlfile?',filename
```

where filename is the name of the XML file created on the server.

Then if the **ComputeProject** return code is **0** and if we run on the Client, we pass **data** as an argument to the Client OWC Spreadsheet **xXMLUrl** property. This results in the Client OWC Spreadsheet downloading the right XML file from the Server and instantaneously populating itself with its content.

As a summary, to use the OWC Spreadsheet on the Server rather than on the client, we have:

- called a function on the Server (**ComputeProject**)

- created another instance of the OWC Spreadsheet in an invisible form on the Server

- done all the necessary work to populate it and format it on the Server

- captured its **xXMLData** property

- created a native file on the Server and filled it with the OWC Spreadsheet **xXMLData**

- returned to the Client the URL necessary for the Client to download this native file

- called the Client OWC Spreadsheet **xXMLUrl** property with this URL as an argument to download the file and populate itself

Note that using a result of **r2 document-filename-delete** in the /owc/xmlfile setup results in the native XML file being deleted as soon as it has been received by the Client. This avoids the Server getting cluttered with the XML native files created by people using our application. This is very important: we should never forget that such an application runs on the Internet and that there may be thousands of people using it every day (or more): this would quickly result in tens of thousands of XML files cluttering the Server!

Yes I know: all this may seem a bit complicated at first, but it works and rather efficiently!

The APL+WebComponent Development Cycle

How do you proceed in practice to write an APL+WebComponent application?

Well this depends if you are writing a brand new application or trying to port an existing APL+Win application to the Web. Assume first you are writing a brand new APL+Win application to be published on the Web.

I recommend the following development cycle:

1. design your application interface first, leaving aside as much code as possible which will run on the Server, concentrate on the interface first.

2. write your **AutoStart** function (the one shown in this example should do).

3. write your **Main** function.

4. go very slowly, i.e. write a couple of lines at a time.

5. always check that you are using APL constructs which are supported by JSaveSDK.

6. test it within the APL+Win ActiveX Server workspace, in APL mode (example: Main''); correct any APL bug ;

7. save the workspace.

8. then go to JSaveSDK and Republish your application
   (once you have selected the functions you want to publish, you need to click 2 buttons in JSaveSDK in this order each time: **Republish** and **Package All**).

9. test your application in the local browser
   (http://localhost:port/application.htm
      example: http://localhost:4000/owc.htm).

10. if everything runs fine, come back to the workspace and write a couple more lines of code and then loop at step 5 ...

11. if a problem occured while testing in the browser, you need to debug the APL+WebComponent version of your application (this is a little hard, see next paragaph, and that's the reason why you really need to write one or 2 lines of code before testing again in APL and then in the browser).

12. once the whole interface is running fine, i.e. all your published functions are running fine in the Browser, you can start writing code running on the Server.

13. remember to use **RunAtServer** to call any subroutine running on the Server.

14. remember that these subroutines must be monadic and return a result.

15. remember that you should NOT do any interface stuff within the subroutines running on the Server: this is reserved to the Client side (do this interface stuff on the Client instead and pass the resulting necessary data as arguments to the subroutines running on the Server).

16. remember that Server subroutines' arguments and results transfer from the Client to the Server and vice versa: as much as possible keep these arguments and result variables as small as possible.

17. in all cases, go very slowly. Re-publish any time you change anything to a published function and test in the browser.

If you are trying to port an existing APL application, things are more complicated, because you'll be inclined to try to use your existing code as is and to publish it as is to go faster. It's almost sure you'll get some headaches doing that.

I would recommend rewriting the application (at least the code which needs to be published) from scratch in the same workspace with different function names and following the development cycle described above. It is almost certain you'll go faster this way.

Debugging an APL+WebComponent Application

This may be very tricky to do.

First let's forget about debugging stuff on the Server side of your application. Remember – the Server side is pure APL+Win and you know how to debug pure APL programs.

On the Client side it is more complex. The reason is that you do not always get a clear error message pointing you to the error. Sometimes you get no error message, but things do not happen in the browser in the same way as they were happening when testing in APL mode.

Sometimes you get an Internet error but it is not explicit enough to let you know where something bumped. Here is an example:

Sometimes you get an APL Error popping up in the browser, but the reported error is further on in the program than the one that really occurred.

Sometimes things are due to your coding because you forgot about some of the JSaveSDK limitations (always keep at hand the following document and always refer to it: "Description of JSaveSDK supported and unsupported APL features".

But sometimes things are due to bugs in the JSaveSDK translation system (i.e. you do everything right and your application still does not run as expected). Fortunately, there aren't many of these, but as for any software, that may happen.

Let's assume I have made an error in the **frAdd.bnAdd.onClick** handler, as follows:

```
[113]  :case'frAdd.bnAdd.onClick'
[114]     ⍝ Read screen data
[115]     name←'fmProject.frAdd.edName' ⎕wi 'text'
[116]     start←'fmProject.frAdd.edStart' ⎕wi 'text'
[117]     start←(↑¨⎕fi¨(start≠'/')⊂start)[3 1 2]
[118]     start[1]←2000+100|start[1]
[119] ⍝   start←100⊥start                ⍝ correct line
[120]     start←⍳start                   ⍝ Error: left ⊥ argument omitted
[121]     end←'fmProject.frAdd.edEnd' ⎕wi 'text'
[122]     end←(↑¨⎕fi¨(end≠'/')⊂end)[3 1 2]
[123]     end[1]←2000+100|end[1]
[124]     end←100⊥end
[125]     progress←↑⎕fi,'fmProject.frAdd.cbProgress' ⎕wi 'text'
[126]     errmsg←RunAtServer AddTask name start end progress
[127]     :if 0∊⍴errmsg
[128]         'fmProject.frAdd.edName' ⎕wi 'text' ''
[129]         'fmProject.frAdd.edStart' ⎕wi 'text' ''
[130]         'fmProject.frAdd.edEnd' ⎕wi 'text' ''
[131]         'fmProject.frAdd.cbProgress' ⎕wi 'text' ''
[132]         'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#00AA00'
[133]         'fmProject.frAdd.lStatus' ⎕wi 'caption'
                   'Record added to the database!'
[134]     :else
[135]         'fmProject.frAdd.lStatus' ⎕wi 'Δstyle_color' '#FF0000'
[136]         'fmProject.frAdd.lStatus' ⎕wi 'caption' errmsg
[137]     :end
```

I have replaced line **119** by line **120** which contains an obvious error (no left argument to the decode primitive).

If we republish and test/run the application in the browser we get the following error:
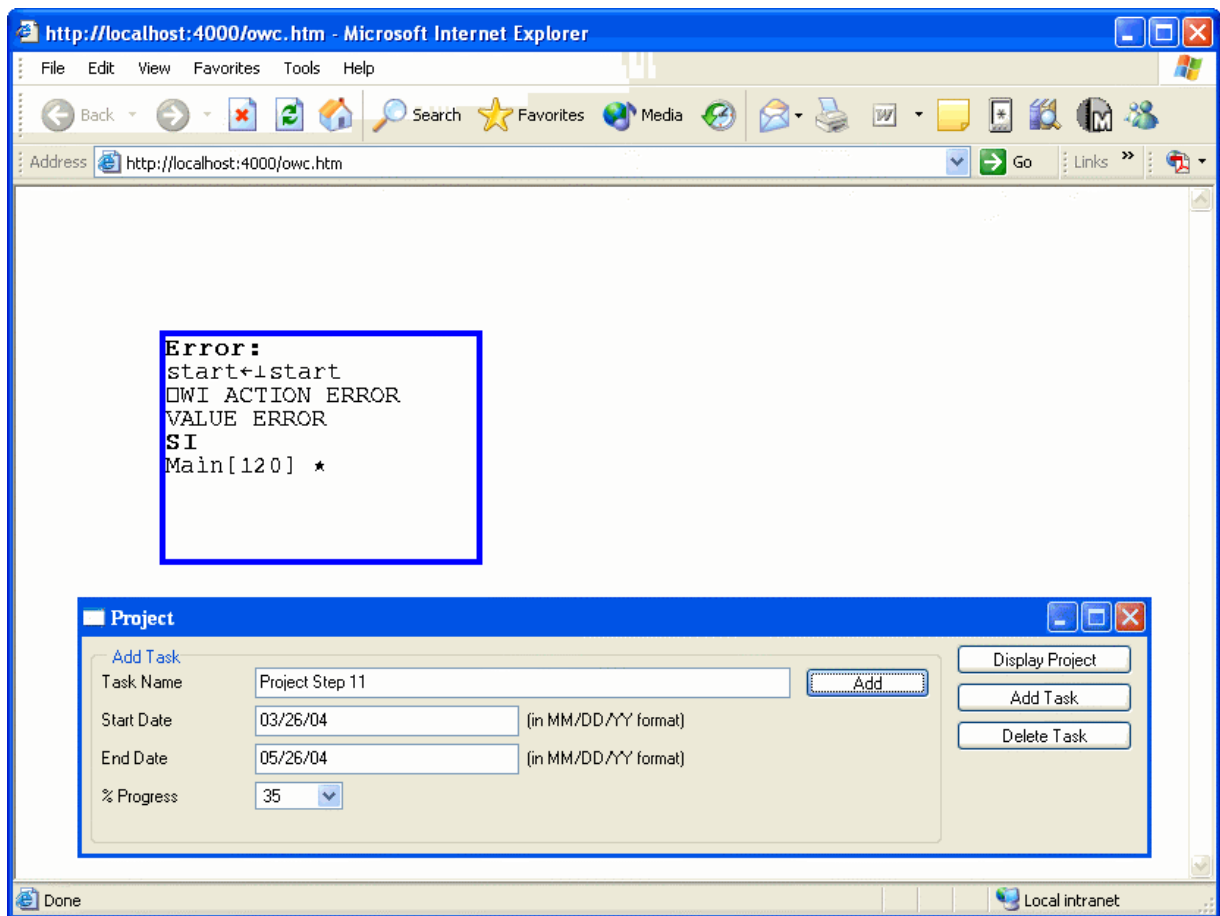
Conclusion

In this second article on APL+WebComponent we have showed a much more sophisticated APL application ported to the Web. If you try this application, please note that it has not been written to handle limit conditions like deleting all tasks, or creating tasks with an end date so far away that it will go beyond the number of columns contained in the OWC Spreadsheet object, etc. If you try the **owc** application, don't try to break it please.

We have explained how to use the APL+WebServicesController to automate setting up your APL+Web Component application, how to develop such an application, how to separate code which needs to run on the Client and code which needs to run on the Server, how to make code run on the Server, how to use the Microsoft OWC Spreadsheet, how to sometimes do interface work on the Server and transfer it to the Client.

We hope you have a better understanding of how to develop APL+WebComponent applications and we hope you will decide to try that soon.

APL Idioms

by Ajay Askoolum

In this article, I raise ten APL problems; each problem has a simple **one line** solution which does *not* involve the creation of any intermediate variable. The objective in solving such problems, for novices and experts alike, is to acquire new APL skills; if you can solve the problem without struggle, find an alternative to the first solution. In short, the solution to the problems should be an idiom: these idioms may be compiled into a Vector idiom dictionary. The solutions will be posted on the *Vector* website (www.vector.org.uk) a week or so before the publication of the *next* issue.

1. How many elements does a given variable have?

If the monadic function CE provides the solution, it should yield the following answers:

Syntax	Answer
CE 90	1
CE 0/8 9 9	0
CE 'ABC' 'DEF'	2
CE 2 1ρ'ABC' 'DEF'	2
CE ''	0

Restriction: assume that the keyboard does *not* have the comma (,) symbol.

2. Is a value within a given range?

For any numeric value and a given range, return a result corresponding to the following table:

Result	Description
2	Value is below the minimum value.
1	Value is equal to the minimum value.
0	Value is between the minimum and maximum value.

¯1	Value is equal to the maximum value.
¯2	Value exceeds maximum value.

Restriction: assume that the keyboard does *not* have any relational operators.

3. Sort a numeric array of integers in ascending order of the number of digits.

For a character array, the solution is simple.

```
      CA←⊃'Sun' 'Mon' 'Tuesday' 'Wed' 'Thursday' 'Fri' 'Sat'
      CA[⍋CA+.=' ';]
Thursday
Tuesday
Sun
Mon
Wed
Fri
Sat
```

Note that the array is sorted in ascending order of the number of spaces in each row and not in any particular alphabetical order. Is this a clue or a red herring? For practice, devise a solution that ignores embedded spaces. For a numeric array, the monadic function SN provides the solution as follows:

```
      SN 8 1⍴89 ¯78 1229 32 129  11 90232 1
     1
    89
   ¯78
    32
    11
   129
  1229
 90232
```

Restriction: assume that the keyboard does *not* have the format (⍕) or quad (⎕) symbol.

4. Return the element(s) of a numeric array indexed by its first dimension.

If the monadic function LD provides the solution, it should yield the following answers:

Syntax	Answer
LD 6	6
LD 98 878 332 2.3 44	44
LD 2 3ρ9 3 4 0.98 22 3.4	0.98 22 3.4
LD 2 3 4ρ+\24/1	13 14 15 16 17 18 19 20 21 22 23 24

Restriction: assume that the keyboard does *not* have the shape of (ρ) symbol.

5. Return the sum of element(s) of a numeric array on its first dimension.

If the monadic function LS provides the solution, it should yield the following answers:

Syntax	Answer
LS 6	6
LS 98 878 332 2.3 44	1354.3
LS 2 3ρ9 3 4 0.98 22 3.4	9.98 25 7.4
LS 2 3 4ρ+\24/1	14 16 18 20 22 24 26 28 30 32 34 36

Restriction: assume that the keyboard does *not* have the plus (+) symbol. I have used +\24/1 in order to ensure that I get the first 24 numbers in index origin 1: you can set ⎕io←1 and use ι24 instead.

6. Convert the string representation of integers to numbers.

If the monadic function CN provides the solution, it should work as follows:

``` CN '898'``` ```898```	``` 10×CN '898'``` ```8980```
``` CN ¨'898' '34'``` ```898 34```	``` 10×CN ¨'898' '34'``` ```8980 340```

Restriction: assume that the keyboard does *not* have the execute (⍎) symbol.

7. Return a numeric array as zeros, increasing the last dimension by 1

If the monadic function ZM provides the solution, it should work as follows:

```
      ZM 2 3 4
0 0 0 0
      ZM 2 3ρι6
 0 0 0 0
 0 0 0 0
      ZM 2 3 4ρι24
 0 0 0 0 0
 0 0 0 0 0
 0 0 0 0 0

 0 0 0 0 0
 0 0 0 0 0
 0 0 0 0 0
```

Restriction: assume that the keyboard does *not* have the times (×), take (↑), minus (−), or comma (,) symbols and it does not end with ρ0.

8. Return the first ones from a Boolean vector.

If the monadic function FO provides the solution, it should work as follows:

```
      FO 2|88 68 45 67 77 90 100 13 27 0
 0 0 1 0 0 0 0 1 0 0
      FO 1 0 1 1 1 0 1 1 0 1
 1 0 1 0 0 0 1 0 0 1
```

Restriction: assume that the keyboard does *not* have the not (~) or rotate (φ) symbols.

9. Return the last ones from a Boolean vector.

If the monadic function LO provides the solution, it should work as follows:

```
      LO 2|88 68 45 67 77 90 100 13 27 0
 0 0 0 0 1 0 0 0 1 0
      LO 1 0 1 1 1 0 1 1 0 1
 1 0 0 0 1 0 0 1 0 1
```

Restriction: assume that the keyboard does *not* have the not (~) or rotate (φ) symbols.

10. Return the elements of a numeric array found at given coordinates.

If the dyadic function RE provides the solution, it should work as follows:

```
      □←A←3 4ρ78 90 22 2.3 43.9 92 12 67 23 33 88 9.34
 78   90 22  2.3
43.9 92 12 67
 23   33 88  9.34
      □←B←2 2ρ1 3 2 4
 1 3
 2 4
     A RE B
22 67


      □←C←2 5 6ρ60?1000
 554 684 485 119 559   530
 193 631 783   1 838   366
 380 987 986 224 269   670
 572 312 314 779 160   285
 168 883 895 230 361   764

 763 891 592  47  51   589
 705 297 689 215 208 1000
 688 772 946 957  16   721
 172 822 982 379 781   163
 797  12 702 723 847   735
     D←⊃(1 3 4) (1 5 5) (2 1 6)
     C RE D
224 361 589
```

Restrictions: assume that a looping solution is not allowed nor one involving semi-colon and that index origin is 1.

If you are a developer working with APL, you should be able to propose at least three solutions or idioms in respect of the problems above. The restrictions imposed in formulating the solution should help in determining an idiom.

Subject to readers' active participation – via correspondence with the editor – the responses will be analysed in a regular feature in future editions of Vector.

Students' Smiles

book review by Cliff Reiter

Mathematical Computing in J, by Howard A. Peelle

Mathematical Computing in J [1] is a smart looking book that very gently introduces mathematics and J to its readers. This book is another J book published by Research Studies Press Ltd., Baldock, Hertfordshire, England; the first was Norman Thomson's *J: the Natural Language for Analytic Computing* [2,3]. *Mathematical Computing in J* is the size of ordinary paper with a soft cover, a wire binding, and a large font. The book is very pleasant to read at the same time as using a computer since it lies flat and is easy to read. The cover is attractive, see below. It is almost 400 pages in length and as we will see, has a remarkably comprehensive style.

Mathematical Computing in J
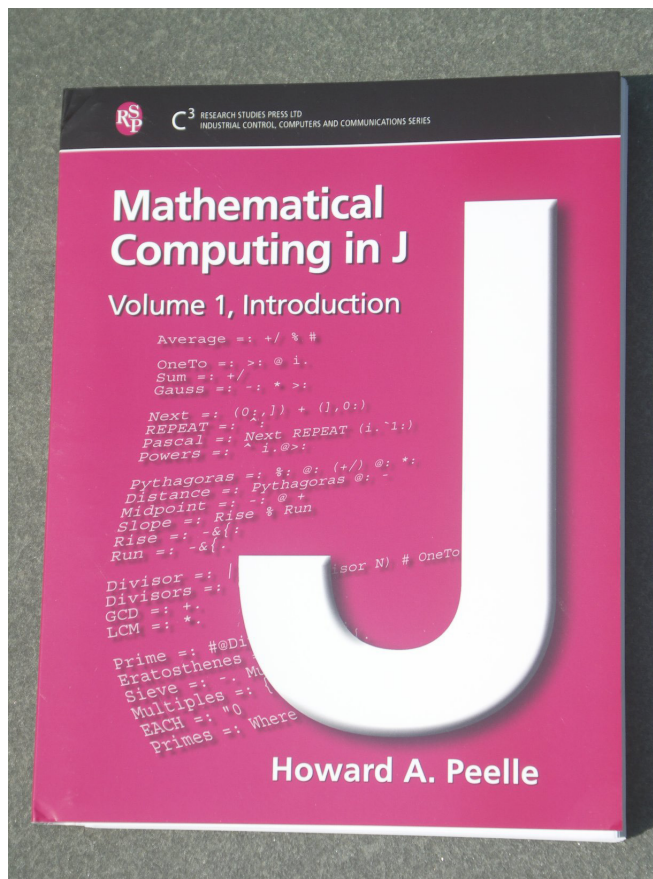
The book is aimed at students and teachers of mathematics or computing at the secondary or early college level. The author says his motivation for writing the book was sparked by students' smiles at understanding by doing. The editor recognizes that being able to implement fundamental mathematical computations is an essential job skill and *Mathematical Computing in J* gives you those skills. Both of their remarks ring true. While the text is in many ways quite straightforward, it is equally brilliant, because the discussion is thorough, careful and motivated.

The topics include those that might be seen by high school: fractions, arithmetic, algebra, equations, exponentials, logarithms, and averaging. Other topics are at a similar level, but might only be seen in high school as enhancement material: Pascal's triangle, moving averages, tables and 3-D arrays, commutativity,

97

associativity, and the sieve of Eratosthenes. We are reviewing the first volume which has 18 chapters; not surprisingly, the second volume will contain more advanced topics including: logic, recursion, probability, statisitics, series, linear algebra, and much more; it is designed to also have 18 chapters.

Each topic in *Mathematical Computing in J* is thoroughly covered. Each chapter is divided into sections: Vocabulary, Worksheet, Explanation, Review and Problems. That is: at the beginning of each chapter there is a vocabulary listing the J introduced in the section and other J that should be reviewed. There is a worksheet intended for interactive completion by the reader. Then there is extensive discussion and explanation of the ideas that the reader should have explored. Possible mistakes or misconceptions are explicitly discussed, important concepts are highlighted, and advanced material is presented, but noted as such.

For example, Chapter 1 is titled: Arithmetic. Division and reciprocal using (%) appear in the experiments that readers should do; in the discussion, the mnemonic that it is similar to the grade school division symbol is mentioned, a warning that it is not (/) is mentioned and consideration of division by zero mentioned, referenced, but not extensively discussed. The reader has been given a comprehensive explanation of the functions, the symbol for the functions and suitable warnings and pointers regarding subtleties. The discussion is followed by a review of J, and then the first problem at the end of the chapter is to calculate the number of feet traveled by a car traveling 55 miles per hour for 3 seconds (1 mile = 5280 feet). While the problem isn't deep, it illustrates an important basic arithmetic computation and exercises the division or reciprocal function. And the reader can't be left behind: the problem solution appears in Appendix 3.

As a second illustration we consider the breadth of the discussion of averaging. Chapter 11 is devoted to averaging and here we consider the worksheet exercises from that section (although in the text, comments appear along with the worksheet experiments, and further experiments are suggested with words; moreover, we show the results of the worksheet, not just the experiments requested).

The worksheet begins with a straightforward example of computing the average of a list of numbers.

1. Averaging Numbers

```
   n =: 85 65 95 90 80

   #n
 5
```

```
   +/ n
415
```

```
   (+/n) % (#n)
83
```

In the next section of the worksheet, the average is computed using a fork, named and unnamed.

2. Functional Averaging

```
   (+/ % #) n
83
```

```
   Average=: +/ % #
```

```
   Average
+-----+-+-+
|+-+-+|%|#|
||+|/|| | |
|+-+-+| | |
+-----+-+-+
```

```
   Average n
83
```

```
   Average 2 4 6 8
5
```

Some weights are defined and used to compute weighted averages.

3. Weighted Average

```
   weights=: 1 0 3 1
```

```
   (+/ weights * 2 4 6 8) % (+/ weights)
5.6
```

```
   n
85 65 95 90 80
```

```
   weights=: 1 2 2 2 1
```

```
   (+/ weights * n) % (+/ weights)
83.125
```

Cumulative sums are given and readers are encouraged to experiment with replacing the sum with averages.

4. Cumulative Sums

```
   +/ \ n
85 150 245 335 415

   +/ \ 1 2 3 4 5
1 3 6 10 15
```

Moving averages are computed.

5. Moving Averages

```
   2 Average \ n
75 80 92.5 85

   2 Average \ 1 2 3 4 5
1.5 2.5 3.5 4.5
```

All the ideas explored in the worksheet are thoroughly discussed in the explanation section. Indeed, the main discussion of forks in the book appears in the discussion section of this chapter. Notice the coherence of the workshop section: several types of averages are discussed. Different styles of computations (direct and tacit) are discussed and monad/dyad cases of (\) are discussed.

Mathematical Computing in J gives a gentle introduction to J in the context of actively doing mathematics. It is a useful, active resource for students learning the mathematical topics being discussed, and is a very gentle introduction to J for those who know the mathematics.

References

[1] Howard A. Peelle, *Mathematical Computing in J, Volume 1*, Research Studies Press, 2004.

[2] Cliff Reiter, Review of *J: the Natural Language for Analytic Computing,* book by Norman Thomson, Vector, 18 3 (2002) 31-37.

[3] Norman Thomson, *J: the Natural Language for Analytic Computing*, J Dictionary (electronic version), Jsoftware Inc., Toronto, 2001.

J-ottings 44: So easy a Child of Ten …

by Norman Thomson (ndt2@tutor.open.ac.uk)

Perfect shuffles just won't go away! Following J-ottings 43 on this subject, Eugene McDonnell, Roger Hui and Jeff Shallit made insightful comments which help cast the problem in a broader context. I shall endeavour to summarise their thoughts here, sauced with generous helpings of J!

To recap, a perfect or ripple shuffle of a deck of cards consists of dividing it into two halves (or as nearly as possible if there is an odd number of cards) and taking one card in turn from each half. A single shuffle of a given number of cards is given by

```
sh=./:@$&0 1          NB. ripple shuffle of i.y.
```

or for repeated shuffling, make the argument into a list (not necessarily numeric)

```
rs=./:0 1&($~)@#      NB. ripple shuffle of y.
```

Assume in what follows that both the word "number" and the letters m, n and k denote "a positive integer", while the letter p means "a prime number" (including 1). A result called Fermat's Little Theorem, first formally proved by Euler in 1736, states that if (n,p) are relatively prime, then $n^{p-1}=1$ in modulo p arithmetic. "Relatively prime" says in words what GCD(m,n)=1 says in maths, or `1=m+.n` says in J, as in the verb:

```
rps=.i.#~(e.&1(+.i.))      NB. relative primes of y.
rps 15
1 2 4 7 8 11 13 14
```

Modulo n arithmetic is what primary school children are familiar with as 'clock arithmetic', that is the arithmetic of a finite set of numbers `i.n` equally spaced around the rim of a clock. A J session can be set up to perform modulo n arithmetic by setting the modulus and defining an adverb such as `mod`:

```
n=.7
mod=.1 : 'n&|@x.'
(6+mod 3),(*:mod 9)  NB. (9 mod 7),(9^2 mod 7)
2 4
```

Advancing a little (but only a little!) beyond primary school, every number possesses a 'totient', where tot(n) is the number of relatively prime numbers which are less than n. Thus tot(2) is 1, tot(3) and tot(4) are both 2 (the relatively

prime number lists being 1,2 and 1,3 respectively), tot(5)=4 (all lower numbers) and so on. tot(n) is often written ϕ(n), and called 'Euler's phi', or in J, `#@rps`. Were this mathematical function just a little more useful, it might well have found a place on calculator keyboards, or indeed as a J primitive, along with factorial, log, sin, etc., and the like. However, it is not necessary to enumerate relatively prime numbers to find tot(n) since it is given by the closed formula

tot(n) = n(1-1/p_1)… (1-1/p_n) where the p's are the unique prime factors of n

Totient can thus be regarded as an extension of `q:` which gives the prime factorisation of n:

```
tot=.*/@,(-.@%)@(~.@q:)   NB. totient (Euler s phi)
(tot 10),(tot 51)
4 32
```

Neither set of parentheses is necessary in the above definition of `tot`, but they help to clarify how it works. `(-.@%)`n is 1 - 1/n, `(~.&.q:)` is the prime factor nub, and the comma makes the hook which multiplies in the factor n.

Euler generalised Fermat's Little Theorem to non-primes by proving that, provided m and n are relatively prime, $m^{tot(n)}$ = 1 (mod n). For primes, *all* preceding numbers are relatively prime, so tot(p) = p-1 and Euler's and Fermat's theorems are equivalent in this case. Some other properties of the totient are simple to prove, viz.

tot(n) is even for all n>2 (this follows from the closed formula)

tot(2^k) = 2^{k-1} (because every odd number less than 2^k is relatively prime)

tot(mn) = tot(m).tot(n) if m,n are relatively prime; and
= n.tot(m) when the prime factors of n are a subset of those of m.

In particular tot(n^2) = n.tot(n). The third of the above properties can be described by saying that tot is a multiplicative function. Generalising the result to prime factor products, if n=(p_1^{k1})(p_2^{k2})…(p_v^{kv}) then

`tot` (n) = `*/` `tot` (p_1^{k1}), `tot` (p_2^{k2}), … ,`tot` (p_v^{kv})

As an aside, the functions tau(n) and sigma(n) as defined below are also multiplicative functions.

```
seldivs=.0&=@|~i.        NB. select divisors of y.
divs=.seldivs~#i.        NB. divisors of y. excl y.
divs 12
1 2 3 4 6
```

```
   tau=.#@,divs              NB. tau=no. of divisors incl y.
   sigma=.+/@,divs           NB. sigma=sum of divisors incl y.
   (tau&>12 13 156);(sigma&>12 13 156)
+------+---------+
|6 2 12|28 14 392|
+------+---------+
```

To illustrate the sort of possible uses for tot(n) and modulo n arithmetic, suppose that the last two digits of 3^{256} (which incidentally has 123 digits altogether) are required. tot(100) = 40 so the problem reduces to that of finding the last two digits of 3^{16} by e.g.

$$(3^{16}) = (81)^4 = (-19)^4 = (361)^2 = 61^2 = 3721 = 21$$

As a further aside, it is not hard to prove that tot(2n) = tot(n) if n is odd and =2.tot(n) if n is even, a result which it is pleasing to have J confirm by comparing matching columns in

```
   (5 6$tot&>>:i.30);5 6$tot&>2*>:i.30
+---------------+----------------+
| 1  1  2  2  4 2| 1  2  2  4  4  4|
| 6  4  6  4 10 4| 6  8  6  8 10  8|
|12  6  8  8 16 6|12 12  8 16 16 12|
|18  8 12 10 22 8|18 16 12 20 22 16|
|20 12 18 12 28 8|20 24 18 24 28 16|
+---------------+----------------+
```

As well as *confirming* results, J can also *suggest* results ahead of proof. For example, the result that tot(3n) = 3tot(n) for multiples of 3, and = 2tot(n) otherwise is forecast with clarity by

```
   (5 6$tot&>>:i.30);5 6$tot&>3*>:i.30
+---------------+----------------+
| 1  1  2  2  4 2| 2  2  6  4  8  6|
| 6  4  6  4 10 4|12  8 18  8 20 12|
|12  6  8  8 16 6|24 12 24 16 32 18|
|18  8 12 10 22 8|36 16 36 20 44 24|
|20 12 18 12 28 8|40 24 54 24 56 24|
+---------------+----------------+
```

Returning to the ripple shuffle problem, the number of shuffles required to restore an even numbered deck of n cards to its original order is the number of times 2 must be multiplied in modulo n-1 arithmetic in order to obtain 1. To obtain such a value, one way is simply to carry on multiplying and reducing modulo (n-1) until 1 is reached, an event which Euler's theorem guarantees is bound to happen. However there may be an earlier arrival at the target than that predicted by Euler's Theorem. For example tot(51) = 32, so that 32 shuffles will restore 52 cards to their original order.

However, if 2 is doubled repeatedly (note a good excuse for a gerund!) :

```
   n=.51                              NB. set modulus
   mod=.1 : 'n&|@x.'                  NB. redefine mod
   p2=.,$:@(+:mod@{:)`}.@.(1&e.) NB. powers of 2
   p2 2
2 4 8 16 32 13 26 1
```

it transpires that a mere 8 steps are sufficient. 8 is called the multiplicative order of 2 (mo2 for short) in modulo 51 arithmetic, and Euler's theorem guarantees that mo2(n) is a divisor of tot(n), which is helpful in manual searches. mo2 is of course just `#p2` . As an alternative to redefining `p2` every time the modulus is reset, write

```
  mo2=.3 :0       NB. mult order of 2 for odd modulus y.
  r=.2
  while.(1~:y.|r)do.r=.x:2*r end. [ 2^.r
  )
     (mo2 13),(mo2 51)
  12 8
```

Now revisit the ripple shuffle with an even number of cards, for example

```
    sh 10
0 2 4 6 8 1 3 5 7 9
```

It takes only a moment to see that 0 and 9 will remain in place in repeated shuffles, and that the second position will be occupied by successive powers of 2 in modulo 9 arithmetic. The number of shuffles to restore a pack with an even number of cards n is thus mo2(n-1). Eugene pointed out that another way to regard a shuffle such as `sh 10` is as a permutation of `i.10`, which can be expressed using `C.` as a combination of cycles:

```
    C. sh 10
+-+---+-----------+-+
|0|6 3|8 7 5 1 2 4|9|
+-+---+-----------+-+
```

and if shuffling is continued until the original order is restored, the cycles emerge in the columns of these lists read as a matrix:

```
   rs^:(i.6)i.10             NB. all distinct shuffles of 10
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8 1 3 5 7 9
0 4 8 3 7 2 6 1 5 9
0 8 7 6 5 4 3 2 1 9
0 7 5 3 1 8 6 4 2 9
0 5 1 6 2 7 3 8 4 9
```

This demonstrates clearly that if 1 is restored to its original position all the other numbers will obediently follow suit. Since *all* the cycles must return to the start point, the LCM of the lengths of the individual cycles determines the number of perfect shuffles to restore a deck of n cards :

```
    cyclecnt=.(#&>)@(C.@sh)
    cyclecnt 10
 1 2 6 1
    ns=.*./@:cyclecnt    NB. no. of restoring shuffles
    ns 52
 8
```

If n is odd, `C.sh n` is the same as `C.sh n+1` only without the final one-element box:

```
    C. sh 9
 +-+---+-----------+
 |0|6 3|8 7 5 1 2 4|
 +-+---+-----------+
```

Thus ns(n) and ns(n-1) are identical in value to mo2(n-1) so that ns(n) is defined for *all* integers. The LCM of the `cyclecnt` of a product mn is the LCM of the `cyclecnt`s of m and n separately, subject to GCD(m,n)=1. For example:

```
    cyclecnt&.> 11 13 143
 +----+----+-------------+
 |1 10|1 12|1 10 12 60 60|
 +----+----+-------------+
    ns&> 11 13 143           NB. LCM(10,12)=60
 10 12 60
```

Generalising the LCM property, $mo2(n) = */ \ mo2(p_1^{k1}), mo2(p_2^{k2}), \ldots, mo2(p_v^{kv})$ which is identical in form to the analogous expression for `tot` above, only with *. (that is LCM) replacing * (multiply). The relationship between the *notions* of multiply and LCM is emphasised by the closeness of the *notation* in J. `mo2` of course is *not* a multiplicative function – perhaps it should be called an LCM-ic function!

Multiplicative order is a property of all relatively prime numbers less than the modulus. `mo10(n)`, where `mo10` is defined analogously to `mo2`, gives the period length of the recurrence in the decimal representation of %n, for example:

```
    (mo10 13),%13
 6 0.076923076923
```

For shuffles where every third card is picked `ns3` counts the number of shuffles to restore:

```
    sh3=./:@$&0 1 2     NB. shuffle with every 3rd card
    sh3 10
0 3 6 9 1 4 7 2 5 8
    C.sh3 10
+-+-----+----------+
|0|7 2 6|9 8 5 4 1 3|
+-+-----+----------+
    cc3=.(#&>)@C.@sh3
    ns3=.*./@:cc3              NB. #shuffles to restore
    ns3&>10 11 12             NB. .. with 10,11 & 12 cards
6 5 5
```

Analogously with ns, ns3(3n) is identical in value to ns3(3n-1), as shown by :

```
    C.sh3 12
+-+---------+----------+--+
|0|9 5 4 1 3|10 8 2 6 7|11|
+-+---------+----------+--+
    C.sh3 11
+-+---------+----------+
|0|9 5 4 1 3|10 8 2 6 7|
+-+---------+----------+
```

although unlike ns, values of ns3(n) no longer coincide with those of mo3(n). The above procedure can be extended to shuffles with picking at any regular interval, and all the previous discussion on shuffles can be condensed into

```
    shn=./:@$ i.               NB.x. cards, pick each y.th
    nsn=.*./@:(#&>)@C.@shn   NB.#shuffles to restore
    (51 nsn 2),(10 nsn 3)
8 6
```

Multiplicative orders are a more general property than shuffle counts. Here is a table of totents and the first three multiplicative orders of the first few integers:

n:	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
tot	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8
mo2		2	_	4	_	3	_	6	_	10	_	12	_	4	_
mo3			2	4	_	6	2	_	4	5	_	3	6	_	4
mo5				_	2	6	2	6	_	5	2	4	6	_	4
tot^2	1	1	1	2	1	2	2*	2	2	4	2*	4	2	4*	4*

n:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
tot	16	6	18	8	12	10	22	8	20	12	18	12	28	8	30	16
mo2	8	_	18	_	6	_	11	_	20	_	18	_	28	_	5	_
mo3	16	_	18	4	_	5	11	_	20	3	_	6	28	_	30	8
mo5	16	6	9	_	6	5	22	2	_	4	18	6	14	_	3	8
tot^2	8	2	6	4*	4*	4	10	4*	8	4	6	4*	12	4*	8	8*

If mo2(n) is equal in value to tot(n) it is called a "primitive root" of n. Roughly speaking, powers of primitive roots exhaust the full gamut of modulo n integers before repeating. Looking at the second and third rows in the table, 2 is a primitive root of some numbers such as 9 and 13, but not of others such as 7 and 17. The table also shows that 3 and 5 are primitive roots of 7. The final row is the totient of the totient which in the case of primes, is also the number of primitive roots. This is also the case for those non-primes such as 9 which possess primitive roots, other non-primes such as 15 have no primitive roots, and are marked with an asterisk in the final row. There is no general formula for primitive roots, but for small numbers such as those given in the table, they are not hard to find, particularly if a computer with J is at hand. For example, 2 is a primitive root of 13 from the table, and the other three are to be found to be 6, 7 and 11 by observing that

```
     6^mod divs tot n=.13      NB. powers of 6 modulo 13
  6 10 8 9 12
```

does not contain 1, and similarly for 7 and 11. Alternatively use lists to test all the candidate numbers simultaneously:

```
     (<>:i.12)^mod&>divs tot n=.13 NB. find pr. roots of 13
  1  2 3  4  5  6  7  8 9 10 11 12
  1  4 9  3 12 10 10 12 3  9  4  1
  1  8 1 12  8  8  5  5 1 12  5 12
  1  3 3  9  1  9  9  1 9  3  3  1
  1 12 1  1 12 12 12 12 1  1 12  1
```

With a little more code all the primitive roots of primes can be extracted in one go:

```
     t#~-.1 e."1 |:(<t=.rps n)^mod&>divs tot n=.13
  2 6 7 11
     t#~-.1 e."1 |:(<t=.rps n)^mod&>divs tot n=.15
```

(null list)

Although this discussion has led into the beginnings of number theory on the one hand and combinatoric analysis on the other, nevertheless a primary school child with outstanding numerical gifts could well appreciate all the *notions* in this article, if not perhaps the *notation*, and could, with at most the aid of a hand calculator, compute the above table of totients and multiplicative orders. Perhaps it is not a coincidence that the abbreviated form is tot(n)!

Zark Newsletter Extracts

edited by Jonathan Barman

Utility Corner: To E Or Not To E

(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.)

In last issue's **Limbering Up** column, you were asked to define a monadic utility function `ENOFF` (Exponential Notation OFF) that behaves exactly like monadic ⍕ except it never returns its formatted numbers in exponential notation.

APL has a tendency to use exponential notation when the numbers it's displaying are very large or very small. For example:

```
      .0000012345
 0.0000012345
      .00000012345
1.2345E¯7
      1234500000
1234500000
      12345000000
1.2345E10
```

While exponential notation does succeed at expressing number in fewer characters, it does not necessarily improve the clarity of the numbers being displayed.

Here's and example (from Jim Weigang) that shows a display with and without exponential notation

```
      M←(10*¯4+⍳8)∘.×1.234 1.235 0 ¯1.235

      ⎕PP←3

      M
 1.23E¯3 1.24E¯3 0 ¯1.24E¯3
 1.23E¯2 1.24E¯2 0 ¯1.24E¯2
 1.23E¯1 1.24E¯1 0 ¯1.24E¯1
 1.23E0  1.24E0  0 ¯1.24E0
 1.23E1  1.24E1  0 ¯1.24E1
 1.23E2  1.24E2  0 ¯1.24E2
 1.23E3  1.24E3  0 ¯1.24E3
 1.23E4  1.24E4  0 ¯1.24E4
```

```
      ENOFF M
   0.00123      0.00124 0      ¯0.00124
   0.0123       0.0124 0       ¯0.0124
   0.123        0.124 0        ¯0.124
    1.23         1.24 0         ¯1.24
   12.3         12.4 0          ¯12.4
  123          124 0           ¯124
 1234         1235 0          ¯1235
12340        12350 0        ¯12350
```

As you can see, the ENOFF function can make a numeric display more meaningful.

The ENOFF functions we received generally take one of two approaches. In the first, logarithms (⍟) are used to determine the relative magnitude of the numbers. From the magnitude and the current setting of ⎕PP (Print Precision), you can determine the appropriate left argument of dyadic ⍕ that will give the desired result.

The following function illustrates the approach for a scalar (single number) argument.

```
       ∇ R←ENOFF N;D;G;W
  [1]   ⍝ Returns ⍕N for scalar N, but never
  [2]   ⍝ returns exponential notation.
  [3]   ⍝
  [4]    G←1+⌊10⍟|N+N=0
  [5]   ⍝ |G is the number of digits to the
  [6]   ⍝ left of the decimal (if G>0) or
  [7]   ⍝ the number of consecutive zeros to
  [8]   ⍝ the right of the decimal (G≤0).
  [9]   ⍝
  [10]  ⍝ Digits to right of decimal
  [11]   D←0⌈⎕PP-G
  [12]  ⍝
  [13]  ⍝ Required field widths...
  [14]   W←D+1⌈G ⍝ Total no. digits
  [15]  ⍝ Negative sign and decimal point:
  [16]   W←W+(N<0)+D≠0
  [17]  ⍝
  [18]   R←(W,D)⍕N ⍝ Format it
  [19]   →D↓0 ⍝ Done if no decimal point
  [20]  ⍝
  [21]  ⍝ Delete trailing 0s:
  [22]   G←+/∧\'0'=⌽R
  [23]   R←(-G+G=D)↓R ⍝ And solitary point
       ∇
```

For ⎕PP←3, here are some intermediate values of this function's local variables. The values are shown for four different settings of the right argument N:

```
        N     0.001234   12.34   ¯0.01234   0.1

[4]    G        ¯2        2        ¯1        0
[11]   D         5        1         4        3
[14]   W         6        3         5        4
[16]   W         7        4         7        5
[18]   R     0.00123    12.3    ¯0.0123   0.100
[22]   G         0        0         0        2
[23]   R     0.00123    12.3    ¯0.0123   0.1
```

Notice that lines [22] and [23] remove any trailing zeros to the right of the decimal point. They remove the decimal point too if the result is an integer.

The submission from Jim Weigang utilises this approach, and has additional logic to handle numeric arrays of any dimensions:

```
       ∇ R←{P}ENOFF N;B;D;E;F;G;S;T;V;W;X;Z;⎕IO
[1]    ⍝ Behaves like monadic ⍕ but never
[2]    ⍝ returns exponential notation.
[3]    ⍝ Like ⍕, it is sensitive to ⎕PP.
[4]    ⍝ Optional left argument is ⎕PP
[5]    ⍝ surrogate
[6]    ⍝
[7]     ⎕IO←1
[8]    ⍝ Set P from ⎕PP if no left arg:
[9]     ⍎(0=⎕NC'P')/'P←⎕PP'
[10]   ⍝ Empty result for empty arg:
[11]    →(0∊S←⍴N)↓L1
[12]    R←S⍴''
[13]    →0
[14]   ⍝ Make numbers a matrix:
[15]   L1:N←((×/¯1↓S),¯1↑1,S)⍴N
[16]    G←1+⌊10⍟|N+N=0
[17]   ⍝ MG is the number of digits to the
[18]   ⍝ left of the decimal (if G>0) or
[19]   ⍝ the number of consecutive zeros to
[20]   ⍝ the right of the decimal (G≤0)
[21]   ⍝
[22]   ⍝ Compute the appropriate Width and
[23]   ⍝ Digits format for each number:
[24]    D←0⌈P-G ⍝ Digits to rt. of decimal
[25]   ⍝ Required field widths...
[26]   ⍝ One blank to left plus all digits:
[27]    W←1+D+1⌈G
[28]   ⍝ Negative sign and decimal point:
[29]    W←W+(N<0)+D≠0
[30]   ⍝ Shift needed to alogn decimals:
[31]    T←((⍴D)⍴⌈/D)-D
[32]   ⍝ One more if decimal absent in
[33]   ⍝ column that has some decimals:
[34]    T←T+(D=0)∧(⍴D)⍴∨/D>0
[35]   ⍝ Increase width for shift:
[36]    W←W+T
```

```
[37]  ⍝ Make each col have uniform width:
[38]   W←(⍴W)⍴⌈/W
[39]  ⍝ Formatted Matrix shape
[40]   G←(1↑⍴N),+/W[1;]
[41]  ⍝ Adjust field widths to slide the
[42]  ⍝ decimal points into alignment:
[43]   W←W-T-(⍴T)⍴¯1↓0,,T
[44]  ⍝
[45]  ⍝ Format each number:
[46]   R←(,W,[2.5]D)⍕,N
[47]  ⍝ Make it a matrix
[48]   R←G⍴(×/G)↑R
[49]  ⍝
[50]  ⍝ Remove trailing zeros to the
[51]  ⍝ right of the decimal, and delete
[52]  ⍝ excess blank columns:
[53]   V←,⌽R ⍝ Work with reversed vector
[54]  ⍝ A trick to avoid zero partitions:
[55]   V←'. ',V
[56]  ⍝ 1s mark first char of each number:
[57]   F←B>¯1↓0,B←V≠' '
[58]  ⍝ Ignore those without a decimal:
[59]  ⍝ F←F\F ⍴ORRED V='.'
[60]   X←V='.'
[61]   Z←(X∨F)/F
[62]   F←F\(Z/1⌽Z)≤F/X
[63]  ⍝ 1s mark leading (nee trailing) 0s:
[64]  ⍝ T←F ⍴ANDSCAN V='0'
[65]   X←V='0'
[66]   Z←~(T←X≤F)/X
[67]   T←~≠\T\Z≠¯1↓0,Z
[68]  ⍝ Undo the trick:
[69]   T←2↓T
[70]   V←2↓V
[71]  ⍝ 1s mark char just past each group
[72]  ⍝ of 0s:
[73]   D←T<¯1↓0,T
[74]  ⍝ Delete adjacent decimal:
[75]   T←T∨D\'.'=D/V
[76]   T←T∨V=' ' ⍝ Delete blanks, too
[77]   D←G⍴T←~T ⍝ 0s mark stuff to delete
[78]  ⍝ When expanding, put 1 blank
[79]  ⍝ between cols:
[80]   E←(B∨¯1↓0,B←∨≠D)/D
[81]  ⍝ Delete 0s and blanks, insert
[82]  ⍝ minimum blanks:
[83]   V←(⍴E)⍴(,E)\T/V
[84]  ⍝ Strip final blank, undo reversal:
[85]   V←⌽0 ¯1↓V
[86]  ⍝
[87]  ⍝ Restore leading dimensions:
[88]   R←((¯1↓S),¯1↑⍴V)⍴V
     ∇
```

In the second approach, monadic format is immediately applied to the numeric argument, in the hopes that APL may not have chosen to foil us with exponential notation. If there is no exponential notation (no E's), we're done. If there is exponential notation, only those numbers using it need to be reworked. Even then, some useful information can be gleaned from the exponential format of the number.

Again, the following function illustrates the approach for a scalar (single number) argument.

```
        ∇ R←ENOFF N;B;D;I;P;⎕IO
[1]    ⍝ Returns ⍕N for scalar N, but never
[2]    ⍝ retuns exponential notation.
[3]    ⍝
[4]     →('E'∊R←⍕N)↓⎕IO←0
[5]     I←R⍳'E'
[6]     P←⍎(I+1)↓R ⍝ Power
[7]     B←I↑R ⍝ Base
[8]    ⍝ No. decimal places in base:
[9]     D←(ρB)-1+(+/'.¯'∊B)+'0'=¯1↑B
[10]    R←1↓(0⌈D-P)⍕N
        ∇
```

Here are some intermediate values of the local variables when ⎕PP←3, using the same settings of the right argument N illustrated above:

```
        N  0.001234  12.34    ¯0.01234   0.1

[4]    R  1.23E¯3    1.23E1   ¯1.23E¯2  1.0E¯1
[5]    I     4          4        5         3
[6]    P    ¯3          1       ¯2        ¯1
[7]    B   1.23       1.23     ¯1.23      1.0
[9]    D     2          2        2         0
[10]   R  0.00123    12.3     ¯0.0123    0.1
```

Notice that the base and power portions of the number (e.g. 1.23 and ¯3 from 1.23E¯3) are analysed to determine the appropriate left argument to dyadic ⍕. In this simple function, dyadic ⍕ is called with a single number left argument, which always returns a leading blank in its result. In the function below, which was submitted by Bruce Hitchcock, the more typical pairs-of-numbers left argument is used.

```
        ∇ R←ENOFF N;B;C;CD;CM;D;E;EX;I;L;M;P;S;SX;T;U;W;X;Y;⎕IO
[1]    ⍝ Behaves like monadic ⍕ but never
[2]    ⍝ returns exponential notation.
[3]    ⍝ Like ⍕, it is sensitive to ⎕PP
[4]    ⍝
```

```
[5]      →('E'∊R←⍕N)↓0
[6]      ⎕IO←1 ⍝ Origin 1 is fine
[7]      R←,R ⍝ Make it a vector
[8]      T←R≠' ' ⍝ Flag non blanks
[9]   ⍝ Inds of starts of no.s:
[10]     S←(T>¯1↓0,T)/⍳⍴T
[11]     E←(T>1↓T,0)/⍳⍴T ⍝ ... and ends
[12]     Y←⍴L←1+E-S ⍝ Lengths of the no.s
[13]  ⍝ Which ones contain E_s:
[14]     T←+\X←R='E'
[15]     I←(T[E]≠T[S])/⍳Y
[16]  ⍝ Numbers to reformat:
[17]     M←(,N)[I]
[18]  ⍝ Indices of the E_s:
[19]     X←X/⍳⍴X
[20]  ⍝ Starts/ends of these no.s:
[21]     SX←S[I]
[22]     EX←E[I]
[23]  ⍝ Power portion of no. (e.g. ¯5 for
[24]  ⍝ 1.234E¯5
[25]     U←(EX-X)+EX<⍴R ⍝ Lengths
[26]  ⍝ U←∊X+⍳¨U:
[27]     U←U/X-¯1↓0,+\U
[28]     U←U+⍳⍴U
[29]     P←⎕FI R[U] ⍝ Use ⎕FI or ⍎
[30]  ⍝ No. decimal places in base (e.g.
[31]  ⍝ 3 for ¯1.234E5 or 0 for ¯1.0E5):
[32]     T←0,+\R∊'.¯'
[33]     D←X-SX+1+(T[X]-T[SX])+R[X-1]='0'
[34]  ⍝ No. decimal places to show:
[35]     D←0⌈D-P
[36]  ⍝ Total width of number:
[37]     W←(R[SX]='¯')+(1⌈P+1)+(D≠0)+D
[38]  ⍝ Reformat these numbers
[39]     M←(,W,[1.5]D)⍕M
[40]  ⍝ Make room to insert them:
[41]     T←(⍴R)⍴1
[42]     U←EX-SX ⍝ Lengths
[43]  ⍝ U←∊SX+⍳¨U:
[44]     U←U/SX-¯1↓0,+\U
[45]     U←U+⍳⍴U
[46]     T[U]←0
[47]     T[SX]←W
[48]     R←T/R
[49]  ⍝ Update lengths, starts, ends:
[50]     L[I]←W
[51]     E←(+\T)[E]
[52]     S←1+E-L
[53]     SX←S[I]
[54]     EX←E[I]
[55]     U←(EX-X)+EX<⍴R ⍝ Lengths
[56]  ⍝ W←∊(SX-1)+⍳¨W
[57]     W←W/(SX-1)-¯1↓0,+\W
[58]     W←W+⍳⍴W
[59]     R[W]←M ⍝ Insert the new numbers
```

113

```
[60]    →(1≥⍴⍴N)⍴0 ⍝ Exit if vector result
[61]    ⍝
[62]    ⍝ Need at least one blank before and
[63]    ⍝ after each no. (before is fine):
[64]     R←R,' '
[65]    ⍝ Find index of decimal point within
[66]    ⍝ each no. (or 1 beyound end):
[67]     T←+\X←R='.'
[68]     T←T[E]=T[S]
[69]     P←(~T)\X/⍳⍴X
[70]     P[T/⍳Y]←1+T/E
[71]    ⍝ No. digits to left of point:
[72]     M←P-S
[73]    ⍝ ... and to right, including point:
[74]     D←0⌈E-P
[75]     D←D+×D
[76]    ⍝ Largest of each by column:
[77]     T←Y÷C←¯1↑⍴N
[78]     CM←1+⌈⌿(T,C)⍴M ⍝ Plus leading blank
[79]     CD←⌈⌿(T,C)⍴D
[80]    ⍝ Replication vector for expanding:
[81]     T←R≠' '
[82]     T[S-1]←(Y⍴CM)-M
[83]     T[E+1]←T[E+1]+(Y⍴CD)-D
[84]     R←((¯1↓⍴N),+/CM+CD)⍴T/R
        ∇
```

From timings we performed on the two functions above, the second approach seems to be quicker, running in 50% to 75% the time required by the first approach. Since these timings depend on the rank and nature of the numbers, we recommend you perform your own timings if speed is critical to your application.

Ed: There was a small problem verifying that the last two functions (which use monadic format) worked correctly, as the formatting rules in different flavours of APL vary slightly. The original article was evidently developed with APL*PLUS, but I used Dyalog APL 8.2 to reproduce and test the functions. There is no leading space for the first column of a formatted matrix in Dyalog APL, so line `[7]` *in the fourth* `ENOFF` *function had to be amended to* `R←,' ',R`*. Also, Dyalog is much less willing to display exponential format for numbers in a simple vector, so extreme measures were necessary to reproduce the intermediate variables when running the simplified (scalar) example function. I never managed to get* `0.1` *to display as* `1.0E¯1` *and had to put up with* `1E¯1` *or* `1.00E¯1` *instead.*

Limbering Up:
Accumulations

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

A classic: suppose you have two numeric vectors whose elements are in one-to-one correspondence. The first (`ACCT`) is a vector of account numbers; the second (`AMT`) is a vector of dollar amounts. The account numbers in `ACCT` are not distinct; they repeat. What are the distinct account numbers? How many times does each account number occur? What is the sum of the numbers in `AMT` for each distinct account number? What is the percent of each number in `AMT` relative to the total for its account?

The efficient APL algorithms for these problems are well known and will be discussed in the next issue. Your task is to *design* the syntax of one or more utility functions that make the solutions to such problems convenient and intuitive.

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.

Crossword Solution

Solution to Crossword in 21.3

A	■	A	P	L	■	■	V			S	■	P
R	■	V	←	v	/	S	<u>∈</u>	C	I	■	O
E	X	■	A	D	φ	M	B	T	■	P	W
A	↑	1	■	V	,	−	V	■	^	\	E
O	K	?	B	■	F	X	■	−	/	M	R
F	■	ρ	V	1	■	■	O	⌈	K	■	S
C	■	V	ι	2	×	⌈	/	N	V	■	O
I	F	■	1	↓	ρ	M	A	T	■	×	F
R	⏀	B	■	V	,	÷	B	■	S	/	F
C	O	P	Y	■	N	L	■	1	∈	G	I
L	■	/	ι	ρ	■	■	C	~	S	■	V
E	X	P	O	N	E	N	T	I	A	T	E

This will be the last in the series, unless there is a storm of protest from readers!

PROFIT

Developer Productivity
with APLX version 3 and SQL

by Ajay Askoolum

MicroAPL have released version 3.0 of their APLX interpreter; this version offers major (unique) enhancements in the way APL can handle data in the workspace and manage its transfer and acquisition to and from other applications/resources available in the host environment.

All the enhancements are exposed using new ⎕ functions and new objects for the ⎕wi function. This may seem like pandering to the APL developer who does not want to stray beyond the comfort zone of the workspace. However, APLX is a cross-platform interpreter and ⎕ functions, where the interpreter transparently manages the internal differences, are a clean way of providing consistent facilities.

Should the developer choose to build for a single platform, say, Windows, APLX can also harness platform-specific features such as Win32 APIs. The deployment of APIs not only makes an application robust (and endows APLX with behaviour consistent with other applications that deploy the same APIs) but also saves the overhead of writing, testing, and maintaining APL code for the same purpose.

Code re-use with ⎕na

Consider the following task: programmatically verify the existence of an Access database and if it does not exist, create it – bear in mind that an MDB file has a custom internal structure. The application expects a database named C:\MYLOC\QTR1\MYDB.MDB. The likely scenarios are as follows:

- One or more levels in the path tree, and therefore the file, do not exist.
- The path exists but the file does not.
- Neither the path nor the file exists.

A pure APL solution will not only be verbose but also require the Access application (used as a COM server) to create the file if it does not exist. A solution based on APIs is simpler – in this instance, neither the path nor the file exists.

```
        ∇CreateMDB
[1]    File←'C:\MYLOC\QTR1\MYDB.MDB' ⍝ File to create
[2]    Path←(⌽∨\'\'=⌽R)/R
[3]    'MakeSureDirectoryPathExists' ⎕na
          'I4 imagehlp|MakeSureDirectoryPathExists <CT[*]'
[4]    0 0⍴MakeSureDirectoryPathExists Path
[5]    'PathFileExists' ⎕na 'I4 shlwapi|PathFileExistsA <CT[*]'
[6]    :If ~×PathFileExists File
[7]       'SQLConfigDataSource' ⎕na
             'odbccp32|SQLConfigDataSource U4 U2 <CT[*] <CT[*]'
[8]       SQLConfigDataSource 0 4
             'Microsoft Access Driver (*.mdb)'('CREATE_DB=',File)
[9]    :EndIf
        ∇  2005-05-07 9.12.09
```

The Internet and the APLX manuals, respectively, document and clarify the deployment of Win32 APIs. After running this function, it would be possible to verify the existence of the database using the PathFileExists function. A more robust test is to verify whether Access can open the file.

```
        'AC' ⎕wi 'Create' 'Access.Application'
        'AC' ⎕wi 'XOpenCurrentDataBase' 'C:\MYLOC\QTR1\MYDB.MDB'
        'AC' ⎕wi 'XCurrentProject.FullName'
C:\MYLOC\QTR1\MYDB.MDB
        'AC' ⎕wi 'Quit'
```

Access does indeed recognise the new file. This demonstrates how close APL comes to application development by the collation of existing building blocks.

Charts using ⎕chart

In his book, *Les APL Étendus* (Masson, 1994, ISBN 2-225-84579-4), B. Legrand discusses the area and perimeter of 2-D surfaces using a chart example. In a single expression, APLX draws the surface, as shown in Figure 1.

```
  'type=line' 'x=First' 'title=Les APL Étendus' ⎕chart ⊃x y
```

This function provides a powerful tool for visualising data on demand: for example, a menu option can read any highlighted data from any source – the grid object, a text file, or an HTML page in the browser object – and provide a chart, and allow the user complete control on its presentation via the standard menu.

The x-y coordinates of the surface are:

```
x←0 2  2  1 ¯1 ¯1 ¯2 ¯1 0
y←2 0 ¯2 ¯3 ¯1  0  1  2 2
```

The menu options allow further dynamic refinements and the saving of the chart as a free-standing file. Even Excel requires multiple steps and programming to achieve the same result!
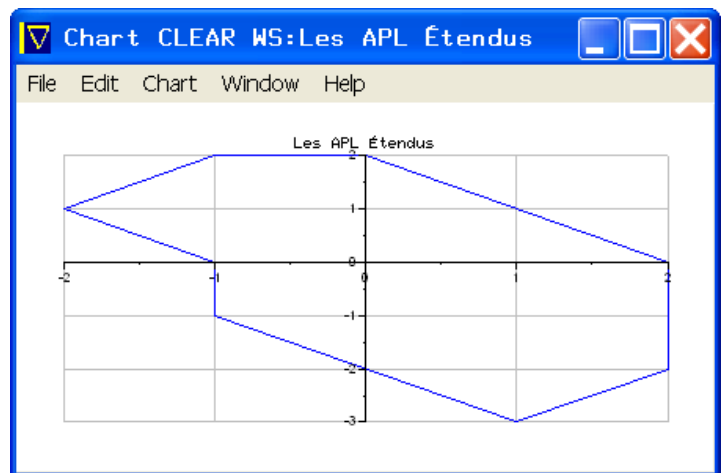
Figure 1. Legrand's Chart

This is a powerful demonstration of APLX's ability to enhance programmer productivity. Besides the system function `⎕chart` there is a chart and a series object, accessible via the standard `⎕wi` function, for more sophisticated graphical representation of workspace data.

Structured Query Language using `⎕sql`

This system function provides a platform independent means of accessing any ODBC-compliant data source (USER/SYSTEM Data Source Names (DSNs) or DSN-less connections are supported but not provider connections.) It takes an optional left-hand argument, and returns a 3-element nested result; the right-hand argument varies depending on the first element of the right-hand argument.

It is not as flexible/powerful as raw ODBC API calls or as ActiveX Data Object (ADO). However, `⎕sql` is much simpler to use and fits with traditional APL programmer expectations: all the code and data is in the workspace.

I would recommend that the optional-left-hand argument is always specified, and that DSN-less connections be used for the following reasons:

> The left-hand argument is an integer that represents the handle of an ODBC connection. Should multiple concurrent connections be necessary, it is advisable to specify the left-hand argument: it provides the context for `⎕sql` operations. The default tie number is 0 and both positive and negative numbers may be used.

> Both USER (accessible only by the user who creates it) and SYSTEM (accessible by any user of the computer on which it exists) DSNs are stored in the Windows Registry: besides the overhead of reading the Registry, such DSNs are not easily distributable with an application. FILE DSNs (held in the filing system and therefore easily distributable) are not supported.

In order to be able to associate the left-hand argument with a data source, you might consider using an implicit numbering convention such as the one listed in Table 1.

Left-Hand Argument	Data Source
16	Text files, including CSV files.
32	Excel workbooks
64	Access databases
128	Oracle databases
256	SQL Server databases
512	DB2 databases

Table 1. Making tie numbers intuitive

This numbering scheme bestows the tie numbers with clues as to their respective data sources; should multiple tie numbers be required, increment the base tie number by one (subject to not exceeding the base number for the next data source). This may be especially helpful when an application supports several data sources: given the tie number, the help desk will know what data source is involved. If this convention is used, the following function will return the association of tie number to data source (note: the function does not verify whether the elements of its right-hand argument actually exist as tie numbers):

```
      ∇Z←L Tie R
[1]   (Z L)←R(R≠0)
[2]   Z←L/Z
[3]   Z←('Text' 'Excel' 'Access' 'Oracle' 'SQL Server' 'DB2' 'Unknown')
         [16 32 64 128 256 512ι2*⌊2⍟|Z]
[4]   Z←L\Z
[5]   ((~L)/Z)←⊂'Unknown'
[6]   Z←(⊃,¨R),⊃Z ⍝ TIP: ,¨ turns simple vectors into nested vectors
      ∇  2005-05-22 9.08.34

      ⎕sql 'Connections'
  129 258 0 ¯512 513 512

      Tie (⎕sql 'Connections'),8982
  129 Oracle
  258 SQL Server
    0 Unknown
¯512 DB2
  513 DB2
  512 DB2
 8982 Unknown
```

Typical DSN-less Connection Strings

Table 2 lists typical connection strings for the data sources considered:

Data Source	Connection String
Text	Driver={Microsoft Text Driver (*.txt; *.csv)};DBQ=C:\;
Excel	Driver={Microsoft Excel Driver (*.xls)};DBQ=C:\APLX.XLS;
Access	Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\APLX.MDB;
Oracle	Driver={Oracle in OraHome9};Server=D2K1Z01J;Database=hr;UID=@;PWD=#;
SQL Server	Driver={SQL Server};Server=D2K1Z01J;Database=pubs;UID=@;PWD=#;
DB2	Driver={IBM DB2 ODBC DRIVER};UID= @;PWD= #;DBALIAS=SAMPLE;

Table 2. Connection strings

Some aspects of the connection strings need clarification:

Parts of connection string will vary depending on your setup; @ denotes the user ID, # denotes the password; if either or both of these parameters are blank, they can be omitted, alternatively, use =; as their respective values.

For the Text driver, DBQ denotes the location but for the Excel and Access drivers, it denotes a fully qualified file name.

For Oracle, HR is a supplied database as is PUBS for SQL Server.

For DB2, SAMPLE is an alias for the default TOOLSDB database created during setup; DBALIAS= is a hybrid of the Microsoft DBQ= and Oracle Database= parameters.

For Oracle, the driver name is likely to vary, depending on the version installed.

For Oracle and SQL Server, the server name will vary.

For MSDE2000 (found on the Office 2000 Professional CD) and SQLExpress (the beta version is freely downloadable subject to end user license agreement from Microsoft) use the SQL Server connection string. These work with SQL Server databases but have restrictions on size and the number of concurrent users; they can be used for development and the databases can be upgraded to SQL Server.

On the Windows platform, ODBC compliant data sources include:

Text files, including CSV files; fixed and ragged edge records are supported.

Excel workbooks.

File based databases such as Access.

Server based databases such as Oracle, SQL Server, DB2, etc.

⎕sql can enumerate the DB2 databases that are available: for DB2, the dedicated syntax for connection does not rely on the specification of a driver.

```
      ⎕sql 'DataSources' 'aplxdb2'
 0 0 0 0   TOOLSDB
           SAMPLE


      ⎕sql 'DataSources' 'aplxodbc'
 0 0 0 0   MQIS                   SQL Server
           Visio Database Samples Microsoft Access Driver (*.MDB)
           MS Access Database     Microsoft Access Driver (*.mdb)
           Excel Files            Microsoft Excel Driver (*.xls)
           dBASE Files            Microsoft dBase Driver (*.dbf)
           LocalServer            SQL Server
           ORACLE9                Oracle in OraHome9
           Text Driver            Microsoft Text Driver (*.txt; *.csv)
           MYACCESS               Microsoft Access Driver (*.mdb)
           IBM                    IBM DB2 ODBC DRIVER
```

It can also enumerate the ODBC drivers installed on a computer. On Linux and MacOS platforms, use unixODBC and iODBC drivers and data sources.

Connecting to the Data Source

Without a connection to an underlying data source, ⎕sql is not of much use. Table 3 lists its parameters.

Parameters	Value	Notes
First	'Connect'	Always the same string. This is not as senseless as it might first appear.
Second	'aplxodbc' or 'aplxdb2'	MicroAPL use aplxodbc for ODBC data sources and aplxdb2 for DB2: Either may be used for DB2, depending on the third argument.
Third	Connection String or DSN name	The connections string and DSN may include the additional UID and PWD parameters.
[Fourth]	Other value	Optional. If necessary, specify UID.
[Fifth]	Other value	Optional. If necessary, specify PWD.

Table 3 Parameters

Should the parameters specified be incomplete, `□sql` invokes the ODBC administrator dialogues to request additional information in order to establish a successful connection.

DB2 can connect either way

Although MicroAPL have suggested the use of 'aplxdb2' for connection to DB2, 'aplxodbc' may also be used to access DB2 databases. This function demonstrates that it is possible to connect to DB2 databases using either 'aplxodbc' or 'aplxdb2':

```
      ∇IBM
[1]   ⍝ Disconnect from all sources
[2]    0 0ρ∈(□sql 'Connections')□sql¨⊂'Disconnect'
[3]   ⍝ Connect using aplxodbc
[4]    0 0ρ∈512 □sql 'Connect' 'aplxodbc'
       'Driver={IBM DB2 ODBC DRIVER};UID= ;PWD= ;DBALIAS=SAMPLE;'
[5]   ⍝ Connect using  aplxdb2
[6]    0 0ρ∈513 □sql 'Connect' 'aplxdb2' 'SAMPLE'
[7]   (RC RM DataODBC)←512 □sql 'Do'
        "SELECT * FROM EMP_ACT where PROJNO LIKE 'IF%' AND ACTNO<100;"
[8]   (RC RM DataDB2)←513 □sql 'Do'
        "SELECT * FROM EMP_ACT where PROJNO LIKE 'IF%' AND ACTNO<100;"
      ∇  2005-05-22 8.24.05
```

The two results – one based on 'aplxodbc' and the other on 'aplxdb2' – are identical. The data is returned as a nested matrix as shown in Figure 2.

The variables RC and RM return debugging information; in this specific instance, I know that there is nothing to debug as the expressions have worked.

Line [7] shows how to record the return values from `□sql`.

The connection syntax is simpler, as seen in line [6], when using 'aplxdb2'; however, the syntax in line [4] is more generic.

All the records resulting from the SQL is returned to the workspace: runtime workspace full errors are likely when processing a large number of records. On the other hand, as the data is in the workspace, it can easily be presented in the grid object. Moreover, if APLX is used to query databases interactively, it is preferable to have all the data in the workspace.

```
┌→
↓ ┌──────┐ ┌──────┐              ┌──────────┐ ┌──────────┐
│ │000030│ │IF1000│ 10 0.5       │1982-06-01│ │1983-01-01│
│ └──────┘ └──────┘              └──────────┘ └──────────┘
│ ┌──────┐ ┌──────┐              ┌──────────┐ ┌──────────┐
│ │000030│ │IF2000│ 10 0.5       │1982-01-01│ │1983-01-01│
│ └──────┘ └──────┘              └──────────┘ └──────────┘
│ ┌──────┐ ┌──────┐              ┌──────────┐ ┌──────────┐
│ │000130│ │IF1000│ 90 1         │1982-01-01│ │1982-10-01│
│ └──────┘ └──────┘              └──────────┘ └──────────┘
│ ┌──────┐ ┌──────┐              ┌──────────┐ ┌──────────┐
│ │000140│ │IF1000│ 90 0.5       │1982-10-01│ │1983-01-01│
│ └──────┘ └──────┘              └──────────┘ └──────────┘
└ε
```

Figure 1 ⎕sql result

```
      DataODBC≡DataDB2
1
      ≡DataDB2  ⍝ I expected it to be 3!
2
      ≡DataDB2[1;1]
2
      ≡DataDB2[1;3]
0
```

There is an inconsistency: numeric values are simple scalars and strings are nested vectors. This has the potential of causing unexpected runtime errors.

Date values are returned as string in ISO 8601 format, that is, YYYY-MM-DD HH:MM:SS.

```
      ρ⊃DataDB2[1;5]
 ⍝ This is a Date
10
      DataDB2[1;5]
 1982-06-01

      ≡¨DataODBC
1 1 0 0 1 1
1 1 0 0 1 1
1 1 0 0 1 1
1 1 0 0 1 1
```

It is easy to cope with a consistent YYYY-MM-DD format: either use SQL or APL code for the transformation into any other format. In order to provide consistency between the representation of dates retrieved from a database and those acquired from the user interface – which would be in the local regional format – some transformation will be necessary.

In contrast, ActiveX Data Object (ADO) returns dates as scalars, which have to be converted with code or SQL: it may be messy if the reference date is unknown or variable as it would be when several data sources are used. The following function extracts the same data as the function IBM above:

```
      ∇Z←ADO;Cns;Sql
[1]   Cns←'Driver={IBM DB2 ODBC DRIVER};UID= ;PWD= ;DBALIAS=SAMPLE;'
[2]   Sql←"SELECT * FROM EMP_ACT WHERE PROJNO LIKE 'IF%' AND ACTNO<100"
[3]   0 0ρ'ADORS' ⎕wi 'Create' 'ADODB.RecordSet'
[4]   0 0ρ'ADORS' ⎕wi 'XOpen' Sql Cns
[5]   Z←'ADORS' ⎕wi 'XGetRows'
[6]   'ADORS' ⎕wi 'XClose'
[7]   'ADORS' ⎕wi 'Delete'
      ∇  2005-05-17 17.57.31
```

The result is shown in Figure 3.

Note the need for transformation (⍉) to present the data in row-major tabulation and, as stated, the dates are scalars.

In order to present the dates in the user interface and for data arithmetic, it is necessary to convert the scalars to a date format. The conversion must use the same date reference as that used by the tool used to retrieve the records, in this case ADO.

Figure 3. ADO result

The ⎕sql dates correspond to the scalar dates upon conversion: this is seen by sending the data to Excel and applying the date format:

```
      ∇Z←ADO2;Cns;Sql
[1]   Cns←'Driver={IBM DB2 ODBC DRIVER};UID= ;PWD= ;DBALIAS=SAMPLE;'
[2]   Sql←"SELECT * FROM EMP_ACT WHERE PROJNO LIKE 'IF%' AND ACTNO<100"
[3]   0 0ρ'ADORS' ⎕wi 'Create' 'ADODB.RecordSet'
[4]   0 0ρ'ADORS' ⎕wi 'XOpen' Sql Cns
[5]   0 0ρ'xl' ⎕wi 'Create' 'Excel.Application'
[6]   0 0ρ'xl' ⎕wi 'XWorkbooks.Add'
[7]   'xl' ⎕wi 'Range().Value' 'A1:F4'(⍉'ADORS' ⎕wi 'GetRows')
[8]   'xl' ⎕wi 'Range().NumberFormat' 'E1:F4' "yyyy-mm-dd"
[9]   'xl' ⎕wi 'visible' 1
[10]  'ADORS' ⎕wi 'XClose'
[11]  'ADORS' ⎕wi 'Delete'
[12]  'xl' ⎕wi 'Delete'
      ∇  2005-05-21 13.14.29
```

Figure 4. Results in Excel

Figures 2, 3 and 4 show the same set of records.

Use the APLX built-in grid object to present the results of ⎕sql in the user interface, where necessary, because:

Additional steps to re-format data will not be necessary.

Data will remain as retrieved from the data source – note that column A is numeric in Excel but is character in the data source.

Working with Tables

A database contains tables (a collection of records), which in turn, contain columns. Although it is convenient to visualize tables as the ubiquitous Excel worksheet, tables are different in several respects:

> Tables contain raw data – there is no data formatting.
>
> Records and columns do not have any relationships except those defined by constraints.
>
> The database manages the addition and deletion of records: the developer is oblivious of the exact position or sequence of records.
>
> The deletion of values in a column does not affect other values; for example, values in adjacent columns do not shift.

The developer will need to query whether a particular table exists, or enumerate the tables that exist, and the columns that each table has. ⎕sql provides intrinsic facilities for such queries. The syntax is:

```
[Tie] ⎕sql 'Tables' [Catalog] [Schema] [Table] [Types]
```

Beneath the surface, ⎕sql is managing a very complex SQL statement that is database vendor specific.

> The first two optional arguments, Catalog and Schema respectively define the owner and the table space: from the point of view of the developer, these can be ignored, usually. Different vendors have different names for the terms adopted by MicroAPL.

The third argument, Table, is the placeholder for a specific table name or, if omitted, for all table names.

The fourth optional argument, Types, allows the developer to enumerate subsets of tables that exist within the database.

During the maintenance cycle of an application, that is, when the application has acquired a legacy in the field, it is may be necessary to add new tables to existing databases. The application needs to check for the existence of a table before it creates it.

Does the table EMPLOYEES exist in the HR database in ORACLE?

```
      ⎕display 128 ⎕sql 'Tables' '' '' 'Employees' ''
┌→────────────────────────────────────────────────────────┐
↓ ┌→──────┐  ┌⊖┐ ┌→──────────────────────────────────────┐ │
│ │0 0 0 0│  │ │ ↓ ┌⊖┐ ┌→─┐ ┌→────────┐ ┌→──────┐ ┌⊖┐     │ │
│ └~──────┘  └─┘ │ │0│ │OE│ │EMPLOYEES│ │SYNONYM│ │0│     │ │
│                │ └~┘ └──┘ └─────────┘ └───────┘ └~┘     │ │
│                │ ┌⊖┐ ┌→─┐ ┌→────────┐ ┌→────┐   ┌⊖┐     │ │
│                │ │0│ │HR│ │EMPLOYEES│ │TABLE│   │0│     │ │
│                │ └~┘ └──┘ └─────────┘ └─────┘   └~┘     │ │
│                └∊──────────────────────────────────────┘ │
└∊────────────────────────────────────────────────────────┘
```
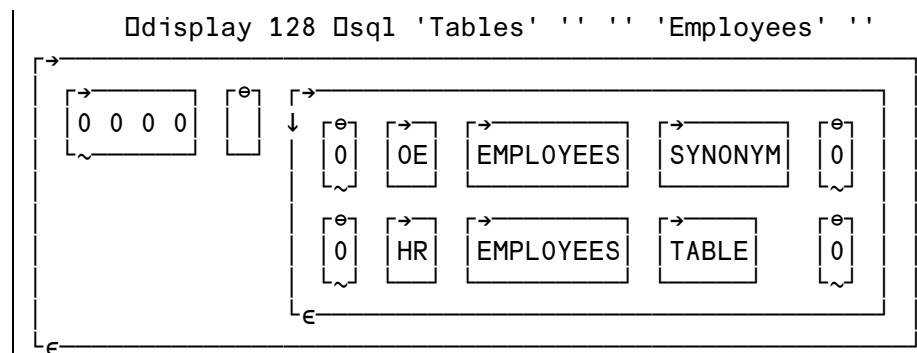
Yes, there are two tables, one each in the OE and HR schemas of the ORACLE connection.

Does a table EMPLOYEES of type TABLE exist?

```
      ⎕display 128 ⎕sql 'Tables' '' '' 'Employees' 'TABLE'
┌→──────────────────────────────────────────────┐
↓ ┌→──────┐  ┌⊖┐ ┌→────────────────────────────┐ │
│ │0 0 0 0│  │ │ ↓ ┌⊖┐ ┌→─┐ ┌→────────┐ ┌→────┐ ┌⊖┐ │ │
│ └~──────┘  └─┘ │ │0│ │HR│ │EMPLOYEES│ │TABLE│ │0│ │ │
│                │ └~┘ └──┘ └─────────┘ └─────┘ └~┘ │ │
│                └∊──────────────────────────────┘ │
└∊──────────────────────────────────────────────┘
```
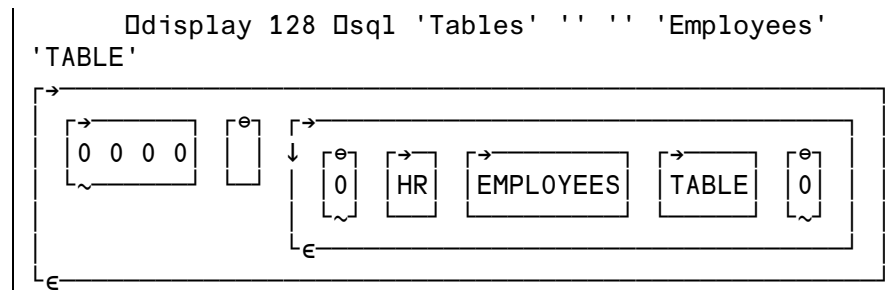
The previous two examples use the SQL tie number 128, defined using the connection string given above. In the latter example, if the first dimension of the third element is zero, the table does not exist.

It may also be necessary to add new columns to existing tables. APLX provides a means of doing so:

```
[Tie] ⎕SQL 'Columns' [Catalog] [Schema] [Table] [Column]
```

What columns exist in the AUTHORS table in the SQL SERVER PUBS database? This is using a different SQL tie number – 256.

```
    2↓256 ⎕sql 'Columns' '' '' 'AUTHORS' ''
pubs dbo authors au_id    12 id       11 11       0                 12   11  1 NO  39
pubs dbo authors au_lname 12 varchar 40 40       0                 12   40  2 NO  39
pubs dbo authors au_fname 12 varchar 20 20       0                 12   20  3 NO  39
pubs dbo authors phone     1 char    12 12       0    ('UNKNOWN')   1   12  4 NO  47
pubs dbo authors address  12 varchar 40 40       1                 12   40  5 YES 39
pubs dbo authors city     12 varchar 20 20       1                 12   20  6 YES 39
pubs dbo authors state     1 char     2  2       1                  1    2  7 YES 39
pubs dbo authors zip       1 char     5  5       1                  1    5  8 YES 39
pubs dbo authors contract ¯7 bit      1  1  0    0                 ¯7        9 NO  50
```

Note that all the details relating to the definition of the table is returned. This is just short of creating the actual SQL script for the creation of the table:

```
CREATE TABLE [authors] (
     [au_id] [id] NOT NULL ,
     [au_lname] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
     [au_fname] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
     [phone] [char] (12) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL CONSTRAINT
[DF__authors__phone__78B3EFCA] DEFAULT ('UNKNOWN'),
     [address] [varchar] (40) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
     [city] [varchar] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
     [state] [char] (2) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
     [zip] [char] (5) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
     [contract] [bit] NOT NULL ,
     CONSTRAINT [UPKCL_auidind] PRIMARY KEY  CLUSTERED
     (
         [au_id]
     )  ON [PRIMARY] ,
      CHECK ([au_id] like '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]'),
      CHECK ([zip] like '[0-9][0-9][0-9][0-9][0-9]')
) ON [PRIMARY]
GO
```

This script was created by SQLDMO; it contains default details that are normally omitted from an SQL statement – note the correspondence between the script and what ⎕sql returns. The provision of the scripts for tables must be on the wish list of the next release: scripts would enable a hassle free distribution of applications during the maintenance cycle.

Does a column STATE exist in the table AUTHORS in the PUBS database?

```
     256 ⎕sql 'Columns' '' '' 'AUTHORS' 'STATE'
 0 0 0 0    pubs dbo authors state 1 char 2 2     1     1    2 7 YES 39
```

Yes, it does. Does a column named APLX exist in the same table?

```
     ρ3⊃256 ⎕sql 'Columns' '' '' 'AUTHORS' 'aplx'
 0 19
```

```
     ρ3⊃256 ⎕sql 'Columns' '' '' 'AUTHORS' 'STATE'
 1 19
```

The first dimension of the third element of the result is zero if the column does not exist.

Working with SQL

SQL is an acronym for Structured Query Language, not Standard Query Language: that means that each driver exposes its own SQL engine. Although all SQL engines comply with most of the SQL92 standard, they also have vendor specific enhancements that are exclusive, inconsistent, or completely different, that is, elusive.

- The Microsoft drivers allow a number of Visual Basic/Visual Basic for Applications keywords as standard.

- The Microsoft T-SQL (SQL Server) and Oracle drivers allow the replacement of null values within the SQL statement but they use different keywords, ISNULL and NVL, respectively.

- Remember to use algebraic rather than APL relational operators within SQL statements; that is use <> instead of ≠ 'not equal to', etc.

- SQL has a null data type (a null value is not equal to any other value, not even another null value) that has no corresponding equivalent in APL. Use IS NULL or IS NOT NULL instead of = NULL or <> NULL.

- SQL has its own conventions: use uppercase, enclose embedded strings in single quotes, and end with a semi-colon. However, SQL statements are not case-sensitive.

Although SQL presents exciting opportunities for APL software development, it presents new challenges especially for applications designed to support several data sources – for example, one client may choose SQL Server while another opts for Oracle.

A generic SQL reference, preferably one that compares SQL dialects is a new requirement for APLX developers, that is, in addition to the APLX documentation. The reference should cover the following:

- The Data Query Language (DQL) component of SQL deals with the retrieval of data from a database – it is based on a single command SELECT. This is the most widely-used component of SQL. Although there is a single command, it is probably the most complex of all SQL commands that developers would deploy routinely.

- The Data Manipulation Language (DML) component of SQL deals with the modification of existing data in a database. The primary commands include INSERT, UPDATE, and DELETE.

- The Data Definition Language (DDL) component of SQL deals with the modification of the structure of existing databases and the creation of new databases. The primary commands include CREATE TABLE, CREATE DATABASE, ALTER TABLE, DROP TABLE, CREATE INDEX, and DROP INDEX.

- The Data Control Language (DCL) deals with database access permissions and used by database administrators who have unrestricted access. The commands in this component of SQL are ALTER PASSWORD, GRANT, REVOKE, and CREATE SYNONYM.

In addition to the SQL reference, database vendor specific references are required for working with databases. In essence, some aspects of data management are held in the data-tier (database) itself rather than in the business-tier (application code). Such references should include the following:

- Constraints: These define the relationship of columns in a table and among tables. For example, there may be such constraints as a column requiring a value (cannot be null) or being unique etc. Constraints reinforce data integrity and affect the way SQL acts on the data source.

- Triggers: These define event driven code that is run, triggered, under some circumstances. For example, a trigger may add a timestamp to a record whenever it is altered.

- Stored Procedures: In simplistic terms, these are complex SQL statements that are stored in and run from the database itself. Stored procedures are efficient because they are optimised by the SQL engine before being stored but SQL statements are optimised on demand.

- Indexes: These are metrics that are calculated and held in the database in order to speed up data access.

Usually, file based databases do not support triggers and stored procedures.

For a robust deployment of a relational database as the data-tier, a third reference is required – one on transaction processing, record locking and multi-user concurrent access management. This is likely to be vendor specific.

Text Files as Relational Tables

A large number of APL applications rely on text files for incoming data. With the Microsoft text driver, `⎕sql` can treat text files as relational tables – strictly speaking, they are not relational tables. This means that text file data can be manipulated without APL code, usually with recourse to native file functions, and data conversion routines are not necessary as the text driver returns data in the

type that is appropriate. In order to promote robust handling of text files, the following is highly recommended:

Include column headers in the first row of data.

As the driver scans a default number of rows in order to determine the data type for the column, the initial rows should contain representative data as far as *type* is concerned.

Learn to create the file SCHEMA.INI: this provides a greater degree of control on how the data is handled.

Consider the example shown in Figure 5:

The file contains the types of data routinely encountered in an APL application.

There are placeholders for missing values: see the last row in Figure 5.

For a robust understanding of how to work with text data files and setup their DSNs, research SCHEMA.INI on the Internet.
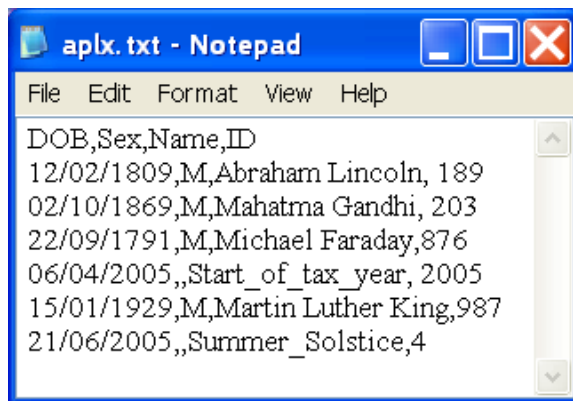
Figure 5. Text data source

Let us connect to the data source; note that the DBQ parameter specifies the location of the actual data source and not the name of the data source itself.

```
16 ⎕sql 'Connect' 'aplxodbc'
   'Driver={Microsoft Text Driver (*.txt; *.csv)};DBQ=C:\;'
```

Some examples illustrating the power of SQL statements are shown below. Although APL coding can achieve the results of the SQL statements, SQL is far more efficient and has the least overhead.

a. Select the records where the column SEX contains missing values:

```
      16 ⎕sql 'Do' 'Select * FROM APLX.TXT WHERE SEX IS NULL;'
 0 0 0 0    2005-04-06 00:00:00    Start_of_tax_year 2005
            2005-06-21 00:00:00    Summer_Solstice      4
```

This query finds two records.

b. Select the records where ID is in the range 100 to 300:

```
      ⎕display 16 ⎕sql 'Do' 'SELECT * FROM APLX.TXT WHERE ID BETWEEN 100 AND
  300;'
```

```
┌→──────────────────────────────────────────────────────────────────────┐
│ ┌→──────┐  ┌⊖┐ ┌→─────────────────────────────────────────────────────┐│
│ │0 0 0 0│  │ │ ↓ ┌→────────────────────┐ ┌→┐ ┌→─────────────────┐      ││
│ └~──────┘  └─┘ │ │1809-02-12 00:00:00  │ │M│ │Abraham Lincoln   │ 189  ││
│               │ └─────────────────────┘ └─┘ └──────────────────┘      ││
│               │ ┌→────────────────────┐ ┌→┐ ┌→─────────────────┐      ││
│               │ │1869-10-02 00:00:00  │ │M│ │Mahatma Gandhi    │ 203  ││
│               │ └─────────────────────┘ └─┘ └──────────────────┘      ││
│               └∊──────────────────────────────────────────────────────┘│
└∊───────────────────────────────────────────────────────────────────────┘
```

c. Refine the previous query by sorting the records in descending order of NAME.

This illustrates the power of SQL statements and indicates the scope that exists for avoiding APL coding to manipulate source data.

```
      ⎕display 16 ⎕sql 'Do' 'SELECT * FROM APLX.TXT WHERE ID BETWEEN 100 AND
  300 ORDER BY NAME DESC;'
```

```
┌→──────────────────────────────────────────────────────────────────────┐
│ ┌→──────┐  ┌⊖┐ ┌→─────────────────────────────────────────────────────┐│
│ │0 0 0 0│  │ │ ↓ ┌→────────────────────┐ ┌→┐ ┌→─────────────────┐      ││
│ └~──────┘  └─┘ │ │1869-10-02 00:00:00  │ │M│ │Mahatma Gandhi    │ 203  ││
│               │ └─────────────────────┘ └─┘ └──────────────────┘      ││
│               │ ┌→────────────────────┐ ┌→┐ ┌→─────────────────┐      ││
│               │ │1809-02-12 00:00:00  │ │M│ │Abraham Lincoln   │ 189  ││
│               │ └─────────────────────┘ └─┘ └──────────────────┘      ││
│               └∊──────────────────────────────────────────────────────┘│
└∊───────────────────────────────────────────────────────────────────────┘
```

Data retain their expected type in the workspace:

```
      (RC RM Data)←16 ⎕sql 'Do' 'SELECT * FROM APLX.TXT WHERE ID BETWEEN 100
  AND 300;'
      Data[1;4]×100
18900
```

d. Add an arbitrary column and make NAME uppercase:

```
      16 ⎕sql 'Do' "Select 'APLX_Demo' as MYVAR,UCASE(NAME) FROM APLX.TXT
  WHERE ID NOT BETWEEN 100 AND 300 ORDER BY NAME DESC;"
   0 0 0 0      APLX_Demo SUMMER_SOLSTICE
               APLX_Demo START_OF_TAX_YEAR
               APLX_Demo MICHAEL FARADAY
               APLX_Demo MARTIN LUTHER KING
```

SQL is far more efficient than the traditional approach of using native file functions. However, be wary of the vagaries of SQL dialects: a thorough understanding of SQL dialects will determine the databases that an application decides to support. It is acceptable that an application will contain database specific code: the more numerous the data sources supported, the bigger the overhead.

Excel Workbooks as Relational Data Sources

I'll use C:\APL.TXT as the source for an XLS file: Start Excel, **File | Open**, select
the file, highlight the range A1:D2 and name it MYTAB, and, finally, **File |
SaveAs** an XLS file named APLX.XLS.

```
      32 ⎕sql 'Connect' 'aplxodbc' 'Driver={Microsoft Excel Driver
(*.xls)};DBQ=C:\APLX.XLS;'
      32 ⎕sql 'Tables' '' '' '' ''
 0 0 0 0    C:\APLX    aplx$ SYSTEM TABLE
            C:\APLX    MYTAB TABLE
```

There are two tables: aplx$ is the sheet name and MYTAB is the named range we
defined. Select each table in turn, as follows:

```
      32 ⎕sql 'Do' 'SELECT * FROM MYTAB;'
 0 0 0 0    12/02/1809 M Abraham Lincoln 189
      32 ⎕sql 'Do' 'SELECT * FROM [aplx$];'
 0 0 0 0                        M   Abraham Lincoln      189
                                M   Mahatma Gandhi       203
                                M   Michael Faraday      876
            2005-04-06 00:00:00     Start_of_tax_year   2005
            1929-01-15 00:00:00 M   Martin Luther King   987
            2005-06-21 00:00:00     Summer_Solstice        4
```

Note that a sheet name has $ as a suffix and it must be enclosed in square
brackets: a name relating to a range is specified as it is.

What happened to the DOB field in the first three records? Null values are
returned because the date range handled by the Excel driver (not Excel) starts
01/01/1900 and those dates were earlier than the start date. This is another quirk
of SQL engines.

- Oracle supports dates in the year range ¯4713 to 9999.

- SQL Server has two date data types. The DATETIME type has the range
  01/01/1753 to 31/12/9999 and the SMALLDATETIME type has the
  operational range 01/01/1900 to 06/06/2079.

- Access has a date range between year 100 and 9999.

- Excel uses 01/01/1900 as the default reference date. Optionally, this can be set
  to 01/01/1904. Dates later than 2078 may not be recognised.

- For text data sources, the Excel ranges apply.

- DB2 has a date range of 1 to 9999.

Let us select all individuals born after 01/01/1900 and create a flag to indicate whether an underscore exists in their names. The following function returns the SQL statement:

```
      ∇Z←GetSQL
[1]   Z←''
[2]   Z←Z,"  SELECT NAME, "
[3]   Z←Z,"         FORMAT(DOB) AS DOBIRTH,"
[4]   Z←Z,"         SEX,"
[5]   Z←Z,"         Format(ID,'0####'),"
[6]   Z←Z,"         IIF(INSTR(NAME,'_'),'','REAL NAME') AS FLAG "
[7]   Z←Z,"    FROM [APLX$] "
[8]   Z←Z,"   WHERE DOB > #01 JAN 1900#;"
      ∇  2005-05-15 13.34.33
```

The result is shown below:

```
      ⎕display 32 ⎕sql 'Do' GetSQL
```


- Always construct SQL statements relating to DML as above to keep it readable: there is a vertical column of spaces separating SQL keywords from other text. Use double quotes within APL and reserve single quotes for embedded quotes.

- Note that the embedded INSTR (like ϵ) uses single quotes: in VBA, double quotes would be used.

- Literal dates are specified unambiguously.

- The date field DOB is transformed to UK short date regional format and has been renamed.

- ID is formatted with leading zeros.

- A number of VBA keywords (FORMAT, IIF, INSTR) are used within the SQL statement.

The Browser Object

The new browser object enables any text file created by APLX and other applications to be viewed within the APL GUI session. This includes some custom format files such as XLS, see Figure 6.

Figure 6. Workbook in browser object

The code is:

```
      ∇Browser
[1]   'BR' ⎕wi 'Create'
'Window'('scale' 5)('size' 470 800)
[2]   'BR.Web' ⎕wi 'New'
'Browser'('align' ¯1)
[3] 'BR.Web' ⎕wi 'onReady' "'BR' ⎕wi
'title' ('BR.Web' ⎕wi 'title')"
[4]   'BR.Web' ⎕wi 'Load' 'C:\APLX.XLS'
[5]   0 0ρ⎕WE 'BR'
      ∇  2005-05-15 14.05.20
```

Ideally, an application should construct HTML output and show it in the browser: further enhancements, such as printing and searching, are required as children of the browser object to make this type of facility useful.

Of course, the browser object can show/read web pages: refer to APLX documentation.

Navigating SQL Results

As the result of the 'Do' directive causes the result of SQL statements to be returned to the workspace, the result can be assigned to an APL variable: an application can loop through the result by reference to the first dimension, which is also a count of the number of records returned.

Should the number of records be likely to cause a workspace-full error, ⎕sql offers another means of retrieving SQL results that mitigates this likelihood. Assume that the number of records returned is 5 million thereby guaranteeing workspace full errors. How will ⎕sql cope?

```
      ∇RecordLoop;Cns;Sql;RecordCount
[1]   Cns←'Driver={SQL Server};Server=D2K1Z01J;Database=pubs;UID=sa;PWD=;'
[2]   Sql←"SELECT COUNT(*) FROM AUTHORS WHERE STATE IN('UT','MI');"
[3]  ⍝ TIP: use 0 0ρ∊ to absorb nested return values
[4]   0 0ρ∊256 ⎕sql 'Disconnect' ⍝ Can disconnect arbitrary tie
[5]   0 0ρ∊256 ⎕sql 'Connect' 'aplxodbc' Cns ⍝ I know it will work
[6]   (RC RM Debug)←256 ⎕sql 'Describe'
[7]   (RC RM RecordCount)←256 ⎕sql 'Do' Sql
```

```
[8]    Sql←"SELECT AU_FNAME,AU_LNAME,STATE FROM AUTHORS WHERE STATE IN('UT','MI');"
[9]    0 0ρε256 ⎕sql 'Close' 'ThisSet' ⍝ Can Close arbitrarily
[10]   0 0ρε256 ⎕sql 'Prepare' 'ThisSet' Sql
[11]   0 0ρε256 ⎕sql 'Execute' 'ThisSet'
[12]   (RC RM Fields)←256 ⎕sql 'Describe' 'ThisSet'
[13]   :While 0≠,RecordCount
[14]     (RC RM Data)←256 ⎕sql 'Fetch' 'ThisSet' 1
[15]     Fields[1;]Fix Data
[16] ⍝ Call function to process variables here
[17]     RecordCount←RecordCount-1
[18]   :EndWhile
[19]   Z←256 ⎕sql 'Close' 'ThisSet'
[20]   0 0ρ⎕ex⊃Fields[1;] ⍝ Clear workspace
       ∇   2005-05-22 11.14.58
```

This function uses two more functions:

```
      ∇L Fix R                                ∇R←EnsureName R
[1]   L←EnsureName¨L              [1]    ((R∈' ')/R)←'∆'
[2]   ♠'(',(⍕L),')←',R'                  ∇   2005-05-22 11.12.37
[3]   ⍝TIP �misimpify nested vector using space
      ∇   2005-05-22 13.06.31
```

The function Fix initialises global variables in the workspace: the variables are the column names extracted by the SQL statement. Since some column names may contain spaces in their names, the function EnsureName replaces embedded spaces by ∆. Databases allow column names with embedded spaces and such names are enclosed in square bracket within SQL statements. However, APL does not allow embedded spaces in variable names – the replacement of spaces by ∆ makes the column names valid APL variable names, in most cases.

In line [6], details of the driver in use is captured in the variable Debug. ⊃Debug contains the information shown in the second column of Table 4.

This information is vital when debugging unexpected behaviour.

Description	Value
Data source name	
Driver ODBC version	03.52
DBMS name	Microsoft SQL Server
DBMS version	08.00.0194
Driver name	SQLSRV32.DLL
Driver version	03.85.1117
Database name	Pubs
User name	Dbo
SQL conformance	Core SQL Grammar

Table 4. Connection details

In line [12] the names of the columns returned by the SQL statement are captured in the variable Fields whose first dimension is four: the rows contain names, descriptions, types, and whether the values can be null. The SQL statements in lines [2] and [8] are identical in terms of the conditions for the selection of records:

the SQL statement in line [2] returns a count of the number of records. This is used to loop through the available records: this happens in lines [13] to [18]. The variables are fixed globally in line [15], and in line [16] the application can call any function that will process the variables as required, using global variable names. These names are overwritten at each iteration.

APLX's documentation suggests the use of the return code, RC, in line [14] for managing the loop: if the last element is true, there are more records.

A function such as RecordLoop enables any number of records to be processed without the risk of encountering workspace full errors.

On 'Do' and 'Fetch'

The 'Fetch' predicate behaves exactly like 'Do' when the named result is not followed by an integer, which indicates the number of records to return.

```
        ∇DoFetch;Cns;Sql
 [1]    Cns←'Driver={SQL Server};Server=D2K1Z01J;Database=pubs;UID=sa;PWD=;'
 [2]    Sql←"SELECT AU_FNAME,AU_LNAME,STATE FROM AUTHORS WHERE STATE
IN('UT','MI');"
 [3]    0 0ρ∊256 ⎕sql 'Disconnect' ⍝ Can disconnect arbitrary tie
 [4]    0 0ρ∊256 ⎕sql 'Connect' 'aplxodbc' Cns ⍝ I know it will work
 [5]    (RC RM DO)←256 ⎕sql 'Do' Sql
 [6]    0 0ρ∊256 ⎕sql 'Close' 'ThisSet' ⍝ Can Close arbitrarily
 [7]    0 0ρ∊256 ⎕sql 'Prepare' 'ThisSet' Sql
 [8]    0 0ρ∊256 ⎕sql 'Execute' 'ThisSet'
 [9]    (RC RM FETCH)←256 ⎕sql 'Fetch' 'ThisSet'
 [10]   Z←256 ⎕sql 'Close' 'ThisSet'
        ∇  2005-05-22 13.27.16
```

In line [5], the records are retrieved using the 'Do' predicate and the same records are retrieved using the 'Fetch' predicate in line [9]. The two sets of results are identical:

```
        DO≡FETCH
 1
```

The reason for providing two pathways is mysterious: I would prefer to use 'Do' for SQL statements that do not return records and 'Fetch' when the SQL statements do return records.

The 'Fetch' predicate moves the record pointer to the next available record. If it is necessary to move to a previous record, it is necessary to re-execute the named result set and to use code to move to the required record. I hope that a future enhancement will provide a less arduous mechanism for moving to the first record or to an absolute position.

What is the Purpose of Named Result Sets?

There are several reasons:

> A single connection may have multiple named SQL result sets. The need for multiple result sets is quite common. Consider an application that produces invoices: it may create one named set for the list of customers and another for their orders.
>
> When named result sets are 'Prepare'(d), the SQL engine tokenises the SQL statement: this makes the execution of the SQL more efficient than the execution of SQL statements using 'Do'. This may make a significant difference in the responsiveness of an application.
>
> The SQL statement for a named result set may contain placeholders for parameters that will be substituted at run time, with 'Execute'. Placeholders are denoted by a question mark (?).

Multiple Named Result Sets

Depending on the driver, it may not be possible to create more than one named set. There is no mechanism to establish which driver supports multiple named sets and which do not using ⎕sql: the only clue is the driver's SQL conformance level – see Table 4.

The SQL Server driver does not appear to support multiple sets:

```
        ∇RecordLoop2;Cns
  [1]    Cns←'Driver={SQL Server};Server=D2K1Z01J;Database=pubs;UID=sa;PWD=;'
  [2]    0 0ρε256 ⎕sql 'Disconnect' ⍝ Can disconnect arbitrary tie
  [3]    0 0ρε256 ⎕sql 'Connect' 'aplxodbc' Cns ⍝ I know it will work
  [4]    0 0ρε256 ⎕sql 'Close' 'A1'
  [5]    0 0ρεZ←256 ⎕sql 'Close' 'A2'
  [6]    0 0ρε256 ⎕sql 'Prepare' 'A1' 'SELECT * FROM AUTHORS;'
  [7]    0 0ρε256 ⎕sql 'Execute' 'A1'
  [8]    0 0ρε256 ⎕sql 'Prepare' 'A2' 'SELECT * FROM STORES;'
  [9]    0 0ρε256 ⎕sql 'Execute' 'A2'
  [10]   'At A1'
  [11]   256 ⎕sql 'Fetch' 'A1' 1
  [12]   'At A2'
  [13]   256 ⎕sql 'Fetch' 'A2' 1
  [14]   'At A1'
  [15]   256 ⎕sql 'Fetch' 'A1' 1
  [16]   'At A2'
  [17]   256 ⎕sql 'Fetch' 'A2' 1
        ∇  2005-05-22 14.04.19
```

This function creates two named sets.

```
        RecordLoop2
At A1
 0 0 0 1    172-32-1176 White Johnson 408 496-7223 10932 Bigge Rd. Menlo
Park CA 94025 1
At A2
 3 0 0 0  [Microsoft][ODBC SQL Server Driver]Connection is busy with results
for another hstmt
At A1
 0 0 0 1    213-46-8915 Green Marjorie 415 986-7020 309 63rd St. #411
Oakland CA 94618 1
At A2
 3 0 0 0  [Microsoft][ODBC SQL Server Driver]Connection is busy with results
for another hstmt
```

As shown above, the second named set, A2, is not accessible.

The Oracle driver does support multiple sets. Using RecordLoop2 with an Oracle driver and reading from the EMPLOYEES and JOBS table, the result is:

```
            RecordLoop3
At A1
 0 0 0 1    100 Steven King SKING 515.123.4567 1987-06-17 00:00:00 AD_PRES
24000      90
At A2
 0 0 0 1    AD_PRES President 20000 40000
At A1
 0 0 0 1    101 Neena Kochhar NKOCHHAR 515.123.4568 1989-09-21 00:00:00
AD_VP 17000    100 90
At A2
 0 0 0 1    AD_VP Administration Vice President 15000 30000
```

If it is necessary to have multiple named sets with a data source that does not support them, create different connections for each named set. Obviously, this will use additional resources and is probably less efficient but it does enable an application requiring multiple named sets to support a data source whose driver does not support them. However, there is an interesting twist. Here is the adapted function:

```
        ∇RecordLoop2A;Cns
[1]    Cns←'Driver={SQL Server};Server=D2K1Z01J;Database=pubs;UID=sa;PWD=;'
[2]    ⎕wa
[3]    0 0ρε256 ⎕sql 'Disconnect' ⍝ Can disconnect arbitrary tie
[4]    0 0ρε257 ⎕sql 'Disconnect'
[5]    0 0ρε256 ⎕sql 'Connect' 'aplxodbc' Cns ⍝ I know it will work
[6]    0 0ρε257 ⎕sql 'Connect' 'aplxodbc' Cns ⍝ I know it will work
[7]    ⎕wa
[8]    0 0ρε256 ⎕sql 'Close' 'A1'
[9]    0 0ρεZ←257 ⎕sql 'Close' 'A2'
[10]   0 0ρε256 ⎕sql 'Prepare' 'A1' 'SELECT * FROM AUTHORS;'
[11]   0 0ρε256 ⎕sql 'Execute' 'A1'
[12]   0 0ρε257 ⎕sql 'Prepare' 'A2' 'SELECT * FROM STORES;'
```

```
[13]  0 0ρε257 ⎕sql 'Execute' 'A2'
[14]  'At A1'
[15]  256 ⎕sql 'Fetch' 'A1' 1
[16]  'At A2'
[17]  257 ⎕sql 'Fetch' 'A2' 1
[18]  'At A1'
[19]  256 ⎕sql 'Fetch' 'A1' 1
[20]  'At A2'
[21]  257 ⎕sql 'Fetch' 'A2' 1
      ∇  2005-05-22 14.33.56


           RecordLoop2A
20928230
20928230
At A1
 0 0 0 1    172-32-1176 White Johnson 408 496-7223 10932 Bigge Rd. Menlo
Park CA 94025 1
At A2
 0 0 0 1    6380 Eric the Read Books 788 Catamaugus Ave. Seattle WA 98056
At A1
 0 0 0 1    213-46-8915 Green Marjorie 415 986-7020 309 63rd St. #411
Oakland CA 94618 1
At A2
 0 0 0 1    7066 Barnum's 567 Pasadena Ave. Tustin CA 92789
```

It works!

Note the size of the available workspace returned before and after the two connections: they are identical – that's the twist. Connections appear to use Windows rather than workspace resources.

Dynamic Parameter Substitution

At the start, I created a new Access database; I'll use it to illustrate dynamic parameter substitution. The function is:

```
      ∇Access R
[1]   0 0ρε64 ⎕sql 'Disconnect'
[2]   0 0ρε64 ⎕sql 'Connect' 'aplxodbc' 'Driver={Microsoft Access Driver
(*.mdb)};DBQ=C:\MYLOC\QTR1\MYDB.MDB;'
[3]   :If 0ερ3⊃64 ⎕sql 'Tables' '' '' 'OBJECTS'
[4]     0 0ρε64 ⎕sql 'Do' 'CREATE TABLE OBJECTS(KEYID TEXT (255),NAME TEXT
(50), DOB DATE,CONSTRAINT OBJECTS_PK PRIMARY KEY (KEYID));'
[5]   :EndIf
[6]   0 0ρε64 ⎕sql 'Prepare' 'Expunge' 'DELETE FROM OBJECTS WHERE KEYID=?;'
[7]   0 0ρε64 ⎕sql 'Prepare' 'Insert'
        'INSERT INTO OBJECTS (KEYID) VALUES(?);'
[8]   0 0ρε64 ⎕sql 'Prepare' 'Update'
        'UPDATE OBJECTS SET NAME=?, DOB=? WHERE KEYID=?;'
[9]   0 0ρε64 ⎕sql 'Execute' 'Expunge'(1⊃R)
[10]  0 0ρε64 ⎕sql 'Close' 'Expunge'
```

```
[11]  0 0ρ∈64 ⎕sql 'Execute' 'Insert'(1⊃R)
[12]  0 0ρ∈64 ⎕sql 'Close' 'Insert'
[13]  0 0ρ∈64 ⎕sql 'Execute' 'Update'(2⊃R)(3⊃R)(1⊃R)
[14]  0 0ρ∈64 ⎕sql 'Close' 'Update'
[15]  0 0ρ∈64 ⎕sql 'Prepare' 'Select' 'SELECT * FROM OBJECTS WHERE KEYID=?;'
[16]  0 0ρ∈64 ⎕sql 'Execute' 'Select'(1⊃R)
[17]  (RC RM Data)←64 ⎕sql 'Fetch' 'Select'
[18]  0 0ρ∈64 ⎕sql 'Close' 'Select'
[19]  0 0ρ∈64 ⎕sql 'Disconnect'
      ∇  2005-05-22 17.51.31


      Access '4122' 'APLXv3' '01/06/2005'
```

In line [17], the values inserted in the database are retrieved in the variable Data:

```
      Data
 4122 APLXv3 2005-06-01 00:00:00
```

Note that the date is returned in the standard ISO 8601 format although it was passed in DD/MM/YYYY format.

Dynamic parameters are consumed in the order the placeholders are specified in the original SQL statement: refer to line [13] and compare it to the way the function is called.

The obvious dividend is that workspace variables can be easily written to database tables. In other words, APL can share data with other applications using databases.

Email Management

The new SendMail and GetMail objects enable the management of emails from within the workspace. It is time to download a free trial copy of version 3.0 from MicroAPL's web site and to experiment! Incidentally, MicroAPL are the only vendor who have this automated facility and offer the cheapest industry-strength APL. The documentation for APLX can be downloaded freely.

Finally

The example code in this article is for illustrative purposes only, and does not fully use the return code and message from `⎕sql` in order to make the code robust and assumes index origin 1. In practice, the information returned by the function must be used and, ideally, the code should be index origin independent.

I invite you to start exploring APLX version 3.0 as it adds powerful features to the developer's arsenal of tools.

My wish list for future enhancements is as follows:

- I would like to see sample workspaces that demonstrate and guide users and developers. At present, there are no sample workspaces. This will address the needs of knowledgeable developers and address some minor shortcomings (the lack of examples in places) in the product documentation.

- I would also like to be able to use provider connections. For text, Excel, and Access data sources, the JET provider is far more powerful than ODBC drivers. Such an enhancement should include support for UDL files (as well as file DSNs for ODBC drivers).

- A means of returning all the name result sets belonging to a given `⎕sql` handle is required.

- I would like the `⎕sql` system function extended so that it produces SQL compliant statements for the creation of tables.

- An omission in the APL documentation is a table providing data types, names, and correspondence among the popular data sources – this is bound to be an onerous task, not least because there are too many data sources.

- A property of the grid object that would allow it to receive data from `⎕sql` directly would be very welcome; alternatively, a means of coercing `⎕sql` to send its data to the grid object would be very effective.

In this article, I have demonstrated how APLX can work with server and file databases using the new facilities: in this respect, APLX is currently unique.

I have used the Windows platform as the basis of my exploration: APLX is platform independent. My investigation includes five data sources but not any open source ones: perhaps someone could investigate APLX with such data sources and write up their findings.

Index to Advertisers

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

Submitting Material to Vector

The Vector working group meets towards the end of the month in which Vector appears; we review material for issue n+1 and discuss themes for issues n+2 onwards. Please send the text of submitted articles (hardcopy with diskette as appropriate) to the Vector Working Group via:

> Vector Administration, c/o Gill Smith
> Brook House
> Gilling East
> YORK, YO62 4JJ
> Tel: +44 (0) 1439-788385
> Email: apl385@compuserve.com

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

Camera-ready artwork (e.g. advertisements) and diskettes of 'standard' material (e.g. sustaining members' news) should be sent to Vector Production, Brook House, Gilling East, YORK YO62 4JJ. Please also copy us with all electronically submitted material so that we have early warning of possible problems.

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 ....£110 ❏

Sustaining membership ....£500 ❏

Non-voting UK member (student/OAP/unemployed) ....£10 ❏

PAYMENT – in Sterling or by Visa/Mastercard 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 or Visa number.

I authorise you to debit my Visa/Mastercard 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