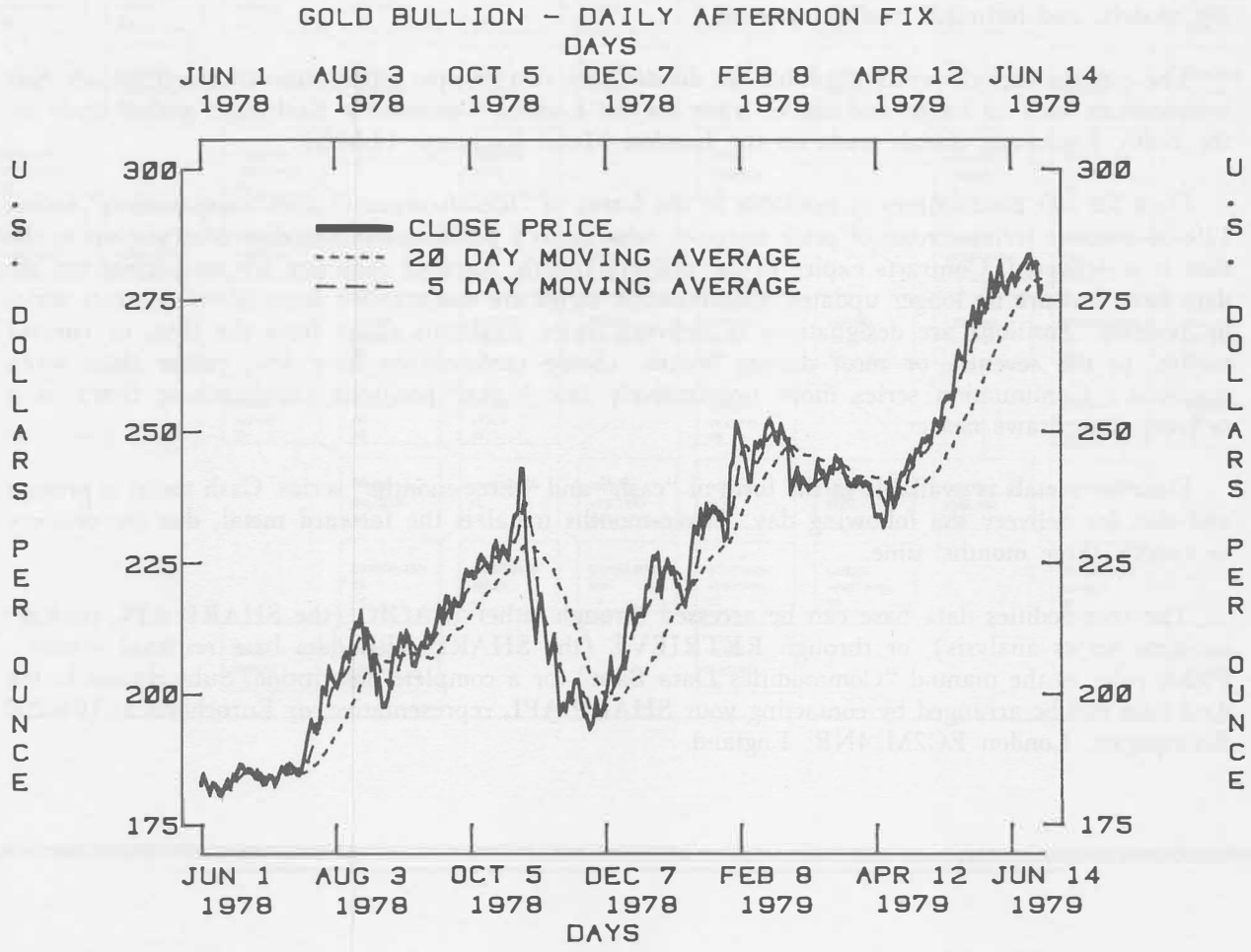
the I.P.Sharp newsletter

July/August 1979
Vol. 7 Number 4

COMMODITY FUTURES DATA NOW ON LINE

Marc Odho, Toronto



The plot above graphically shows the recent price increase of gold bullion. Also shown are the five and twenty day moving averages over the same period. A comparison of the 5-day (short-term) and 20-day (medium term) moving averages indicates a price increase will occur when the short-term moving average becomes greater than the long-term moving average, and vice versa.

The gold bullion fix is an official price made twice a day, morning and afternoon, by a committee of bullion dealers. As British residents cannot deal in gold as such for investment purposes, the London market represents dealing done for trade and other purposes. Non-British clients, banks for instance, also use this market.

IN THIS ISSUE

Technical Supplement 21

Commodity Futures Data

Inflation and fluctuating international currency exchange rates are causing more and more commercial interests and individual investors to hedge and speculate on commodity futures markets. Now current and historical price and volume statistics for all commodities traded on the London Futures market are available on SHARP APL.

The data is supplied and maintained by Eurocommodities Chart Services Limited (Eurocharts). Daily data is available from 1974, and monthly data from 1960.

The data base is suitable for use by financial institutions and by industrial companies that are involved in commodity trading, price behaviour research, economic studies, econometric price forecasting models, and technical trading systems.

The commodities currently available are divided into two groups: soft commodities and metals. Soft commodities such as coffee and cocoa, trade on the London Commodity Exchange; grains trade on the Baltic Exchange; metals trade on the London Metal Exchange (LME).

Data for soft commodities is available in the forms of "life-of-contract" and "continuation" series. Life-of-contract series consist of price statistics relating to a particular commodity contract, up to the date it is delivered. Contracts expire in the delivery month. Expired contracts are maintained on the data base, but are no longer updated. Continuation series are summarized from life-of-contract series by position. Positions are designations of delivery times. Positions range from the first, or current month, to the seventh, or most distant month. (Some commodities have five, rather than seven positions.) Continuation series move progressively into higher positions (approaching first) as a delivery date draws nearer.

Data for metals is available in the form of "cash" and "three-months" series. Cash metal is prompt and due for delivery the following day. Three-months metal is the forward metal, due for delivery in exactly three months' time.

The commodities data base can be accessed through either MAGIC (the SHARP APL package for time series analysis), or through RETRIEVE (the SHARP APL data base retrieval system). Please refer to the manual "Commodities Data Base" for a complete description. Subscription to the data base can be arranged by contacting your SHARP APL representative, or Eurocharts at 194-200 Bishopsgate, London EC2M 4NR, England.

Heard in passing ...

Legislative debate is similar to a user-defined dyadic function:

you can get by without the left,
but you will make no progress without the right argument.

WE ARE PLANNING ANOTHER APL USERS MEETING AND WOULD LIKE YOUR HELP !

I.P. Sharp Associates is sponsoring its second APL Users Meeting, to be held in Toronto from October 6-8, 1980. We are planning for the conference now, and would like you, the APL user, to give us your ideas. If you have any suggestions for speakers to present papers, or sessions that you would like to see added or omitted, we would like to hear your thoughts. Following is a preliminary list of topics for the agenda.

- Budget and Financial Statement Consolidation
- The Use of the Network for Corporate Communications
- Forecasting and Statistical Techniques
- APL as a Language for Decision Support Systems
- APL and Graphics
- Migration to In-House Systems
- Applications in the Aviation Industry
- Data Base Applications in APL
- Managing APL Projects
- Applications in the Actuarial and Insurance Industry
- APL Programming Tools and Techniques.

We also intend to have technical sessions in the evenings; the following four sessions have been planned tentatively and we welcome your suggestions on these as well:

1. How to teach APL
2. Nested Arrays and how to use them
3. Designing systems using Shared Variables
4. Designing interactive systems using Event Trapping.

The APL Users Meeting is a conference for **all** APL users. Whether you are a student, a long-time user of APL on a time-sharing service, someone managing an in-house system, or maybe someone just contemplating the use of APL, you will enjoy meeting with your colleagues, attending a wide variety of sessions and discussing APL applications.

If you have something to contribute to your next APL Users Meeting, please write or phone Rosanne Wild at Sharp's offices in Toronto (416-364-5361). We welcome your comments. For those who did not attend our last meeting in September 1978, copies of the proceedings are still available at $15.00 each.

APL79

The very successful STAPL-sponsored conference has come and gone. Close on 800 people came to Rochester N.Y. to participate and exchange ideas. Proceedings may be ordered prepaid from:

Association for Computing Machinery,
P.O. Box 12105, Church St. Station,
New York, N.Y. 10249.

Prices for the 2-volume set:	ACM and/or STAPL members	$17.00 prepaid
	All others	$22.00 prepaid.

GENESIS — A METAL-MINE MODEL

Del R. Ramage, Economics and Planning Division,
B.C. Ministry of Energy, Mines & Petroleum Resources, Victoria.

The GENESIS valuation model is a private model for evaluating mine property, either on a project basis over the entire life of the mine, or in comparative mode as a tool for sensitivity analysis. The model is particularly suited to analysis of various taxation policies. The major development work to date has been in this area. At present, the model is only appropriate for evaluating single mine operating entities, and will not handle integrated multi-mine corporations. Support for the latter will be incorporated in future enhancements.

The model yields deterministic results, as opposed to a stochastic model which yields a range of results and their associated probabilities of occurrence. Stochastic models are based on manipulation of probabilistic input data through such techniques as Monte Carlo simulation.

In its current form, the model operates on user-input data regarding capital and operating costs, metal prices, mining and milling rates, reserve grades, and other inputs which produce various financial statements and valuation measures. For a given computer run, the user must at present specify (or approximate) all the relevant parameters.

The following valuation measures are produced by GENESIS:

1. Net present value, discounted at three different user-chosen rates;
2. Discounted cash flow rate of return on both total investment before tax, and on equity basis after tax;
3. Undiscounted payback period, and discounted cash flow payback period employing the three discount rates used for 1.

In addition, the following complete annual schedules are produced at the user's option:

1. **Production Schedule** itemizes physical production rates, grade cutoffs used, mill recoveries, inventory levels, etc.

2. **Cashflow Summary** itemizes all sources and uses of funds, yielding annual net cash flows and discounted net cash flows.

3. **Tax Calculation Schedules** presents a tax revenue summary of total Federal and Provincial (B.C.) Corporate income taxes and mineral resources tax paid over the project life, as well as yearly schedules for the above taxes which itemize all major deductions under each tax system.

4. **Income Statement and Effective Tax Rates** presents a detailed yearly income statement itemizing revenues, operating costs, interest charges, depletion, depreciation, and tax liabilities. Dividends are calculated where this is desired. The effective tax rates on an undiscounted and discounted basis for each tax, and for total taxes, is also available.

The GENESIS metal-mine model was designed and programmed in SHARP APL by the Economics and Planning Division of the Ministry of Energy, Mines and Petroleum Resources with I.P. Sharp Associates as prime consultant, and is the property of the Ministry. It is available for use outside the Ministry, but the actual running of the model will be done by Ministry personnel within the Ministry only. The model is in the development stage. As such, we welcome inquiries regarding model operation, and comments and suggestions from users can serve as input to future modifications in operation, scope and application.

SECURITY IN EDP — SHARP SPECIAL SYSTEMS

The design and implementation of minicomputer-based systems for process control, data processing, and for telecommunications inside the company have been organized under the heading **Sharp Special Systems** since 1974.

Recently, Michael Grohn of the Ottawa Special Systems Group, presented a paper, "Sharing Computer Resources Securely", to the seventh annual Canadian conference on Information Science. The conference was held May 12-15 in Banff, Alberta, and attended by about 160 people.

Grohn's paper addressed the problems associated with computer security by guaranteeing controlled access to the central processor. By retrofitting secure front-end and back-end computers to an existing computer system, ("bracketing" the central processor), he provides protection against unauthorized use of the system. The front-end handles data between the user terminals and the system, and the back-end handles the interface between the stored data and the computer system. The front-end is an enhanced communications controller, managing terminal input/output, and providing protection against unauthorized user actions. The back-end is implemented at the functional level of a device controller. It monitors data transfers between memory and mass storage devices, and checks that I/O commands accessing data files represent actions which are consistent with the user's security level. One feature of this scheme is the provision of a direct link, for control purposes (**not** data), between the front-end and back-end processors. The research project is sponsored by the Department of National Defence.

Other papers presented at the conference covered a broad range of topics, with an emphasis on library automation, computer networking, and the use of large data bases. The conference was organized by members of the Canadian Association for Information Science.

AVIATION BULLETIN:

AIR TRANSPORTATION SEMINAR
Newport Beach, California

I.P. Sharp Associates' involvement in the air transportation community has continued to expand. Our L.A. office will host another Air Transportation seminar, to discuss studies and applications in progress throughout the world. Our 1979 seminar will be held on October 11th and 12th, at the Balboa Bay Club in Newport Beach, California.

The two days will be divided into half-day sessions. The seminar has been organized for executives and professionals concerned with all phases of the air transportation industry.

We invite you and other members of your organization to attend and participate. There is no registration fee. Invitations will be mailed out in August, 1979. For more information regarding this seminar please contact your local I.P. Sharp office.

PEGGY KUEFFER MOVES TO NEWPORT BEACH

Peggy has recently joined the Newport Beach office from Reseda (which will be closed). Peggy joined I.P. Sharp Associates two years ago and will continue her responsibilities of coordinating education and marketing of the Sharp Aviation products with the U.S. branch offices.

AVIATION APPLICATIONS LIBRARY

Brian Oliver, Miami

The I.P. Sharp Aviation Applications Library 703 is an organized collection of report programs for the aviation data bases, contributed by I.P. Sharp users and I.P. Sharp staff. The programs fall into seven categories:

	Data Base	Workspace
* FORM41	(traffic and financial schedules)	703 *FORM41*
* ER586	(service segment traffic)	703 *ER586*
* OAND	(origin and destination statistical sample)	703 *OAND*
* OAG	(Official Airline Guide schedules)	703 *OAG*
* B43	(engine and hull inventories)	703 *B43*
* INS	(Immigration and Naturalization statistics)	703 *INS*
* T6	(charter traffic)	703 *T6*

Each workspace contains all the data base access functions contained in its sister workspace in library 702, supplemented by a few utility programs including *SELECT*, which makes the report available in the workspace.

Each report program is fully documented; most of the functions are unlocked and may be modified to suit your special requirements. More than 25 applications programs are currently available in library 703 and the list is growing. All questions and contributions are welcome — direct them either to your I.P. Sharp representative, or to **Peggy Kueffer** in Los Angeles.

An example from library 703 follows. The operating expenses for a specific aircraft type (or group) is printed for a specified calendar year. Amounts are in dollars per block hour — the number of hours that passengers are in the aircraft, including airborne hours and ramp hours.

```
      )LOAD 703 FORM41
SAVED  5.20.56 06/08/79

FOR INSTRUCTIONS, TYPE:   DESCRIBE

      SELECT

PROGRAM NAME? (CARRIAGE RETURN TO EXIT) : ACEXP

MAIN PROGRAM NAME IS : ACEXP

FOR FULL DOCUMENTATION, TYPE:     HOW


      ACEXP
ENTER AIRCRAFT TYPE CODE: (E.G. 8021):
□:
      10
ENTER DATE: (E.G. YEARLY,DATED AT 77):
□:
      YEARLY,DATED AT 78
ENTER CARRIER CODE(S): (E.G. AA,UA):
□:
      AA,UA,CO
```

Technical Supplement–21

OUR RESIDUE BECOMES FUZZY

Eugene E. McDonnell, Palo Alto

A recent paper [1] discusses the reasons for making APL's residue function tolerant, or "fuzzy", and gives an algorithm for making it so. Shortly after 1 September the SHARP APL system will implement a fuzzy residue function. A defined function *FR*, which models the behavior of the new residue function, is given below and may also be found in the temporary workspace 1 *FUZZYRES*:

```
    ∇ Z←A FR W;S
[1]   ⍝MODEL OF FUZZY RESIDUE
[2]   Z←(W×A=0)+((A≠0)∧(⌈S)≠⌊S)×W-A×⌊S←W÷A+A=0
    ∇
```

Users can experiment with it to see what effects the new residue function will have on their work. This note will try to alert users of the system to the differences it will mean to them.

Very broadly speaking, fuzzing residue means that whenever the left argument, or modulus, approximately divides the right argument, the result will be zero. The test whether, in $A|B$, B can be said to be approximately divided by A will be made by asking whether $\lceil B\div A$ is exactly equal to $\lfloor B\div A$, where both the floor and the ceiling functions are fuzzed.

The benefits to the user will be 1) the floor and ceiling functions will now be harmonious with the residue function, and users can expect that such identities as $Z=(\lfloor Z)+1|Z$ will be true in general, and 2) the cases which formerly gave counterintuitive results, such as:

```
      0.2|1 1.2 1.4 1.6 1.8
1.3878E¯17 0.2 0.2 1.1102E¯16 6.9389E¯17
```

will be replaced by:

```
      0.2|1 1.2 1.4 1.6 1.8
0 0 0 0 0
```

Those users who find it important to obtain the behavior of the residue function as it is currently implemented will be able to do so by setting the comparison tolerance $\square CT$ to zero, except that it will not be possible to cause a domain error with any numerical arguments.

The amount by which a function in APL is made fuzzy is not constant, but is related in some way to the argument or arguments of the function. This amount is typically found by multiplying $\square CT$ by the magnitude of the argument, for a monadic function, or by the larger of the magnitudes of the two arguments, for a dyadic function. In the case of residue, it is formed by multiplying $\square CT$ by the magnitude of the quotient of the right argument divided by the modulus. (In all cases, the relative fuzz is constrained to have a minimum value of $\square CT$ and maximum value of 0.5.)

On many APL systems relative fuzz is determined not by the multiplication described above, but by a shortcut approximation method, with the consequence that a strange behavior is seen with certain arguments. You can find out whether your APL system gives accurate results or uses a shortcut by the following test:

```
      A←15.999999999999
      B←15.99999999999989
      C←16
```

As you can see, A is less than B and B is less than C. However, with $\square CT$ set at $1E^{-}13$, many systems will give the following surprising results:

```
      A=B
0
      A=C
1
```

On the SHARP APL system the results will be:

```
      A=B
1
      A=C
1
```

And the behavior on the Sharp system is as it should be, since *A* is within relative fuzz of both *B* and *C*.

Some special cases of residue

There are some aspects of the new behavior of residue which will probably be of less significance to the average user, but which are given in detail here to alert those who may be affected by it.

1. Quotient greater than ÷⎕*CT*

One of the principal effects of introducing fuzz to functions is that it alters the boundary at which one number will be considered a zero with respect to another number. As an example, consider:

```
      1E20=100+1E20
1
```

This result comes about because the 100 loses all significance after its exponent is adjusted to match that of 1*E*20 (this is the way floating point addition takes place in machines using the System 360 architecture). Thus 100 is essentially a zero with respect to addition with 1*E*20. When equality is fuzzed, however, even a number which maintains full significance in addition may be considered a zero with respect to the other number. Thus, we have (with default ⎕*CT*):

```
      1E15=1+1E15
1
```

even though the sum may be represented exactly.

A similar thing arises with fuzzy residue. The present residue accurately gives 1 as the result of 2|1+1*E*15. However, the quotient (1+1*E*15)÷2 is greater than ÷⎕*CT*, and thus the new residue will find that the floor and ceiling of (1+1*E*15)÷2 are equal, and thus will give the result 0. The reasons for the behavior of the floor and ceiling functions are given in [2].

2a. Large modulus and signs of arguments opposite

Another new behavior for residue gets rid of an impossible result. With the current residue, whenever the arguments are opposite in sign, and the modulus is sufficiently much larger than the right argument, the result is identical to the modulus. You can see why this occurs by working through a case such as 1*E*20|¯1. The formula for residue with these arguments is ¯1-1*E*20×⌊¯1÷1*E*20. The floor is exact, and gives ¯1, and thus we get a result exactly equal to 1*E*20. Since the result of residue must always be between zero (inclusive) and the modulus (exclusive), the new residue will give the result zero for these cases.

2b. Large modulus and signs of arguments the same

If *A* divides *B*, then it also divides -*B*. Thus, if we give the result zero for 1*E*20|¯1, we must give the result zero for 1*E*20|1, too. The new residue will be different from the current residue in these cases. The current residue gives:

```
      1E20|1
1
```

but the new residue will give:

```
      1E20|1
0
```

3. Eliminating the domain error (for very small modulus and very large right argument)

There is a last subtle change in the new residue, one that I doubt many of you were aware of at all. The domain error in residue has been removed. What domain error? It's the one you get when you try, for example, `1E¯60|1E60`. Since the quotient is a larger number than can be represented in the machine, the current residue function blows up with a domain error, exactly as if the division `1E60÷1E¯60` had been attempted. The new residue function says that the result in all such cases will be zero, whatever the setting of $\square CT$.

The encode function

Although the encode function is defined in terms of the residue function, it will not be affected by the change to residue. It will still perform as if the comparison tolerance were set to zero. The reason for leaving it unaltered is that it is frequently used for converting a large integer (greater than $+\square CT$) into a vector of numbers into a single large integer (packing). It is desirable that no fuzz be applied to the encode function, in order that such operations continue to be executed will full accuracy.

References

[1] McDonnell, E. E., "Fuzzy Residue", **APL79 Conference Proceedings**, ACM, New York, 1979

[2] Bernecky, R., "Comparison Tolerance", **SHARP APL Technical Notes**, SATN 23, 1978

See also page 8: "How Fuzzy is Fuzz?".

Summer Sum

Gene Mcdonnell, Palo Alto

Historical Note: The Duke of Wellington gained much of his reputation before the battle of Waterloo in the Peninsula War, fought in Portugal and Spain. Among the many sites associated with his name there are the capital cities of Lisbon and Madrid. The puzzle below connects Lisbon and Madrid in a curious way with Wellington, the capital of New Zealand.

Puzzle: Two planes are on the ground at the airport in Wellington, New Zealand. They take off ten minutes apart, and each flies at a constant 1000 km/hr. The first plane flies directly to Lisbon, Portugal, reaching there in just under 19 hours and 35 minutes. Then, continuing on the same course, and at the same rate of speed, it flies on to Madrid, reaching there in just over half an hour. When it arrives, it finds that the second plane has already landed and has been there for more than three minutes. How do you explain this?

Hint: workspace 13 *NATIONS* contains among other things a matrix giving the capitals of the United Nations, and vectors giving their latitudes and longitudes, and two functions, *DIST* and *DIST2* which may be used to find distances between points on the earth, given their latitudes (left argument) and longitudes (right argument). *DIST* assumes that the earth is a sphere 40000 km in circumference, and *DIST2* uses the measurements of the earth as given by modern geographers (it assumes, that is, that the earth is an oblate spheroid).

A list of names can be formed using an expression such as *NAMES←LIST*. The function *LIST* is interactive and uses successive nonempty lines of input to form the rows of a character matrix. An empty line of input causes the function to terminate.

The latitudes of any of the capitals of the United Nations can be found by using a phrase such as *LATS←LATITUDE WHERE CAPITAL IS NAMES*. Similarly, longitudes could be determined by *LONGS←LONGITUDE WHERE CAPITAL IS NAMES*.

The distances between the indicated points could be found by an expression such as *DISTS←LATS DIST2 LONGS*.

The solution to the puzzle will be provided in the next issue.

TEESON'S TEASER

It's very interesting to see how APL people approach problem solving. There is often a remarkable commonality in the initial stages, and then an incredible diversity as further stabs are made. Take, for example, the last teaser. It's a pretty trivial problem but it can lead to a few moments of fun and insight. The problem was to elegantly turn a vector *V* into a 1-column matrix (see technical Supplement 20, page T6 for further details).

Everyone's first approach was

1. `((ρV),1)ρV`

and soon thereafter

2. `(⍴1,ρV)ρV`

or

3. `⍉(1,ρV)ρV`

It seems that there is an aversion to all those parens.

Other solutions in the non-bizarre category were:

4. `0 ¯1 ↓ V,[.5]V`

5. `(V,[⎕IO+.5] ''ρ1↑0ρV)[;,⎕IO]`

A more obscure answer was

6. `V∘.+,0` ⍝ *NUMERIC ONLY*

whilst the bizarrest one was

7. `(V⊤1)/V∘.⌊V` ⍝ *NUMERIC ONLY*

By the way, folks: which, in your opinion, is "the most elegant"? Why?

Here's another 5-minute teaser:

Given a vector, *V*, and an index *I* (`⎕IO←1`) into it, what is the most elegant way to insert another vector, *S*, of the same data type
(a) preceding the *I*'th element
(b) succeeding the *I*'th element?
How would your solution change if `⎕IO←0`?
How would your solution change to be `⎕IO` independent?
How would your solution change to handle differing data types?

If you have a teaser, why not box me with it? Or send your solutions to this one.

PHT.

EVENT TRAPPING

Paul Berry, Palo Alto

The text of this article is taken from the draft of "The SHARP APL Reference Manual", scheduled to appear later this year. Part I appeared in May/June.

Introduction (from Part I)

Automatic Trapping of Errors and Interrupts

SHARP APL now provides ways in which you can state in advance what you want the system to do when execution in your active workspace encounters an error or an interruption. The system may then respond automatically to an event which otherwise would have caused work to be halted.

An interrupt or error which can be intercepted is referred to as a **trappable event**. The term "event" is deliberately vague, partly to reflect the wide variety of things it covers, and partly to allow for the possibility that in the future the category may be expanded. It might then include some events that are neither errors nor interrupts, and do not halt execution.

When you provide an alternate instruction to be executed if a certain event occurs, you are said to **trap** that event. The SHARP APL event-trapping facility consists of the system variables ⎕*ER* and ⎕*TRAP*, and the system function ⎕*SIGNAL*.

Trap definition

The variable ⎕*TRAP* is used to indicate what events you want to trap, and what you want done when they occur. ⎕*TRAP* should be a character array containing any number of trap definitions. Each trap definition is a list of events (identified by number), together with the action to be executed when one of them occurs.

Form of a trap definition: Each trap definition contains at least two fields, and, optionally, a third. The three possible fields are:

1. **Event numbers** list by number the events to which the trap definition applies;
2. **Action code** is a single letter, separated from the adjacent fields by blanks;
3. **Trap line or keyword** (optional after certain actions) instructs the system what it should do following the event, and (optionally) how execution should be resumed.

PART II

Possible actions in the trap definition

Following the field containing the list of event numbers, each trap definition has a second field containing an **action code**. The action code must be separated from the event numbers by at least one blank.

Following certain actions, there may be a third field. When the third field appears, it must be separated from the action code by at least one blank. The third field may be a **trap line** (optional with action *C* or *E*), or a **keyword** (optional with action *D*).

Each action is represented by a single letter. The five possible actions are:

evnums *C* **[line]** CUT
evnums *E* **[line]** EXECUTE
evnums *N* NEXT
evnums *S* STOP
evnums *D* **[keyword]** DO

Event numbers: The first field of any trap definition is a list of the event numbers to which it applies (shown in the examples as **evnums**). The list of event numbers contains one or more numbers, in any order. Each number is represented using the characters 0 through 9. Representations of the various numbers in a list must be separated by blanks. It isn't necessary to use blanks to separate **evnums** either from the delimiter or from the action code, but unneeded blanks are often inserted to make the trap definition easier to read.

evnums *C* line Cut back and execute trap line

When the system detects an event whose number is included in **evnums**, it aborts the execution of all functions invoked later than the function to which the relevant ⎕*TRAP* is local. (**Note:** A function that was invoked later appears higher on the state indicator.)

Every function started later than the function to which the relevant ⎕*TRAP* is local, is aborted. That's true regardless of whether the function was pendent or was suspended.

The system removes the names of the aborted functions from the upper rows of the state indicator. At the same time it removes their line numbers from the leading components of ⎕*LC*. This removal from the state indicator and from ⎕*LC* is referred to as "cutting back the state indicator"; hence the code "*C*."

After the system has cut back the state indicator, it sets the new value of ⎕*ER*. Then it executes the trap line (if any). The trap line may contain any number of statements, separated by diamonds. Its total length may not exceed 500 characters, not counting leading or trailing blanks.

If the trap line contains an error, the system halts; an error in a statement contained directly in the trap definition can't itself be trapped. However, when the trap statement invokes a defined function or uses ⍎ to execute a character vector, errors in that function or in the argument of ⍎, are trappable in the usual way.

Branch with action *C*: When the trap line includes a branch statement, the system resumes execution of the function in which the relevant ⎕*TRAP* was localized (which, since the state indicator has been cut back, is now its top line).

The system understands the value that appears to the right of → in the branch statement to be a reference to a line or label in the function now at the top of the state indicator, just as it would if execution were halted and the branch statement were entered directly from the keyboard. Note that the trap line does **not** show on the state indicator, and is **not** represented in ⎕*LC*, so there's no need to allow for it when you state the destination of the branch.

evnums *E* line Execute trap line

When the system detects an event whose number is included in **evnums**, it immediately sets a new value for ⎕*ER*. Then it executes the trap line (if any). The trap line may contain several statements separated by diamonds. However, its total length may not exceed 500 characters, not counting leading or trailing blanks.

If the trap line contains an error, the system halts; an error in a statement contained directly in the trap definition can't itself be trapped. However, when the trap statement invokes a defined function or uses ⍎ to evaluate a character vector, errors in that function or in the argument of ⍎ are trappable in the usual way.

Branch with action *E*: When the trap line includes a branch statement, the system resumes execution of the function it was working on, at the line indicated. A branch statement appearing in a trap definition is executed exactly as if it had been entered from the keyboard. Note that the trap line does **not** show on the state indicator, and is **not** represented in ⎕*LC*, so there's no need to allow for it when you state the destination of the branch.

Watch out: When the statement following action *E* contains a branch, the system branches to the indicated line **of the function on which it was working when the event occurred.** That may not be the function which set ⎕*TRAP*, but another function invoked subsequently. About the only branch whose effect is certain with action *E* is →⎕*LC*—and even that assumes that any function whose execution may be halted can safely be restarted on any line. If you want to branch to a line other than the line on which the system was working when the event occurred, it is much more prudent to use action *C* rather than *E*.

evnums *N* Skip to next level

When the system detects an event whose number is a member of **evnums**, it abandons the search for trap definitions in the ⎕*TRAP* at which it is now looking. Then it moves to the next level at which ⎕*TRAP* is localized, and continues its search.

What happens after action *N* is what would happen if the ⎕*TRAP* simply made no mention of those events. Action *N* is useful primarily when combined with the inclusive event numbers 0 or 1000 For example, a ⎕*TRAP* containing the two trap definitions

```
⎕TRAP←'∇ 2 14 N ∇ 0 E CORRECTION'
```

has the effect of saying, "For all errors **except** 2 and 14, execute *CORRECTION*."

evnums *S* Stop

When the system detects an event whose number is a member of **evnums**, it halts work and displays an error message, in the same way that it does when you don't use event trapping. It returns to immediate execution.

In effect, action *S* turns off the automatic handling of events. It is useful primarily during debugging, to prevent the system from attempting to deal automatically with errors other than those you anticipate, or to exclude certain errors from a general trap definition, for example:

```
⎕TRAP←'∇ 2 14 S ∇ 0 E CORRECTION'
```

evnums *D* keyword Do

Action *D* is available only with event number 2001; conversely, event 2001 can be followed only by action *D*. Hence, 2001 is the only event number that may appear in the trap definition for action *D*.

In a trap definition, the event number 2001 stands for any return to immediate execution. That does not include ⎕ input mode.

When the system detects a return to immediate execution, it checks to see whether there is a trap definition for event 2001. There might have been a return to immediate execution because a function invoked from the keyboard (or by ⎕*LX*) has reached a normal end. Or it might be because work has been halted following an error or an interrupt for which there is no trap definition, or for which the trap definition doesn't include a branch statement.

When the system returns to immediate execution following an interrupt or error, the system first displays whatever message is called for. **Then** (if it has returned to immediate execution) it checks whether there's a trap definition for event 2001. If there is, it acts on the keyword that follows action code *D* in the trap definition.

Note that because the system handles the error or interrupt first, and then checks 2001, you might see it display an error message which mentions a particular function, but by the time the keyboard is returned to you, that function may no longer exist on the state indicator.

Normal end of execution: When the system reaches a normal end of a function that was invoked from the keyboard (or by automatic execution of ⎕*LX*), **first** it exits from the function, and **then** it checks for a trap on event 2001. A trap for 2001 that was local to the function from which the system just exited will by then no longer exist. Hence, a local trap definition containing action *D* can apply only to a halt other than a normal end.

Effect of keywords with action *D*: When the system finds a trap definition for event 2001, what it does next is determined by the keyword appearing to the right of *D*. The keyword may be empty, or may be *CLEAR* or *EXIT*.

Effect of empty keyword: When the keyword is empty, the system returns to immediate execution in the usual way.

Effect of keyword *EXIT*: The system aborts execution of the most recently invoked function (the one at the top of the state indicator). If that function was invoked during use of ⎕ or ⍞, it aborts the ⎕ or ⍞ as well. It cuts back the state indicator until the top row refers to a defined function (rather than to ⎕ or ⍞). At the same time it makes a similar adjustment to ⎕*LC*.

To the function now at the top of the state indicator, the system again signals event 2001. The new 2001 is treated according to the provision for event 2001 in whatever ⎕*TRAP*s remain after the state indicator has been cut back.

When the trap definition 2001 *D EXIT* to which the system was responding was one it found in a ⎕*TRAP* local to the function just aborted, that ⎕*TRAP* no longer exists; the system searches anew for a trap definition for event 2001. If it finds one, it acts on it. If it doesn't find one, it returns to immediate execution.

The sequence "abort the current function, and again signal event 2001" is repeated indefinitely until the system finds that it has no definition for event 2001, or a definition with an empty keyword.

If at each level the system continues to find 2001 *D EXIT*, the effect is equivalent to)*RESET*.

In that case, the system returns to immediate execution with the state indicator empty, but with no effect on global functions or variables.

Effect of keyword *CLEAR*: The workspace is cleared, as it would be if you'd entered the command)*CLEAR* from the keyboard. You may want to do this either as a rather Draconian security measure, or as a convenience to prevent the system from saving the active workspace following normal end of an N-task or B-task.

Watch out: If you want to clear the workspace only at the normal end of a job (so that the workspace of an N-task or B-task is saved only if it's interrupted), the trap definition 2001 *D CLEAR* should appear only in the global value of ⎕*TRAP*; as long as the function is active, the global value should be superseded by a local ⎕*TRAP* containing a trap definition 2001 *D* (without the keyword *CLEAR*).

The trap definition 2001 *D CLEAR* replaces the function *CLEAROUT* in workspace 1 *WSFNS*, which is now considered obsolete. However, *CLEAROUT* and its companion *NOCLEAR* still work.

***CLEAROUT* takes precedence over all event trapping:** Before the introduction of event trapping, the functions *CLEAROUT* and *NOCLEAR* in workspace 1 *WSFNS* provided a service similar to that now provided by the trap definition 2001 *D CLEAR*. The former facility is now considered obsolete, but still works on the SHARP APL system. However, in a workspace in which *CLEAROUT* has been set, ⎕*TRAP* is completely inoperative. Instead, **any** error or interrupt causes the system to clear the workspace.

Events that can't be trapped

Certain errors and interrupts can't be trapped. They are the following:

1. An error encountered while executing the line that is part of the trap definition. An error in the trap line is exempt from trapping primarily to avoid endless recursions that might otherwise arise from a defective trap line.

However, an error in a function invoked by the trap line, or in the argument of ⍎, **can** be trapped in the usual way.

A function invoked by the trap line but aborted by ⎕*SIGNAL* can also be trapped. In that case, ⎕*ER* reproduces the trap statement.

2. An interrupt caused by ⎕*BOUNCE*. When a task is "bounced," provided *CLEAROUT* has not been set, its active workspace is saved. ⎕*ER* in the saved workspace will show one of the interrupts. Which one depends on the circumstances when the task was bounced.

3. Any of the following errors is considered an error in transmitting or setting up the line for execution, rather than an error in execution. None of them causes ⎕*ER* to be set. None of them has an event number.

OPEN QUOTE ERROR The line contains unbalanced quotes.

CHAR ERROR The line as received from the terminal contains an unrecognizable character. The system displays the characters preceding the first offending character, and awaits a corrected entry.

WS FULL ERROR There is insufficient space to prepare the line for execution. **Note:** This message is distinct from event 1 *WS FULL*, which indicates that there isn't space to complete execution of the line.

SYMBOL TABLE FULL ERROR There is insufficient space in the symbol table to prepare the line for execution. **Note:** This message is distinct from event 15 *SYMBOL TABLE FULL*, which indicates that the symbol table lacks space to accommodate names generated during execution of the line.

DEFN ERROR The line contains an improper use of the ∇ character.

Signalling a user-defined error

The function ⎕*SIGNAL* generates a signal that an error has occurred. By using ⎕*SIGNAL*, a defined function can be made to behave in the same way as a primitive function. That is, when something goes wrong, its execution is aborted, and an error is signalled in the APL statement that invoked it.

Once it's been signalled, the error event produced by ⎕*SIGNAL* can be handled in the same way as errors that the system detects for itself. That is, the event may be handled by a trap definition that includes its event number. Alternatively, if there is no appropriate trap definition, the system halts execution with an error message, and displays the statement that invoked the aborted function.

Building and testing functions that use event trapping

While automatic handling of errors or interrupts offers many advantages, there are ways in which it complicates the task of developing a new application.

If you're used to using APL without event trapping, you're likely to assume that when anything goes wrong, the system will halt with an error message, and you can decide then what steps to take next. You may assume that if work isn't proceeding as you like, you can always halt it by signalling an interrupt, and then take remedial action. But once you set ⎕*TRAP*, an error may produce new behaviour, bypassing the messages on which you previously relied, and resuming execution without halting at all. The situation may be even more perilous if you use ⎕*TRAP* to intercept interrupts signalled from the terminal. You may find it impossible to interfere with a function's execution; you may find that setting a stop vector (with *S*∆) no longer has the expected effect. Of course, it is precisely to produce such invulnerability that you may wish to trap errors. The risk is that doing it incorrectly may produce some very inconvenient behaviour.

Suggested precautions: When you use event trapping, you may need to plan and test your programs with a thoroughness and caution that would not be needed without ⎕*TRAP*. The following are some rules of thumb that may be useful.

1. Test the trap line by entering it from the keyboard. The trap line which accompanies actions *C* or *E* is executed in the same way as if work halted and you entered it from the keyboard. You can exploit this fact when you're building a new application. Start with trap definitions which include the event number and action code *E* or *C*, but which provide no trap line, or a trap line consisting solely of *⎕ER*.

During a trial run of your function, when the event is intercepted, the system will halt and will return control to you at the keyboard. That will permit you to look around before committing yourself to the next step. When you have settled on a sequence of statements which, when entered from the keyboard, produce the behaviour you want, make them the trap line of your trap definition.

2. Include a trace capability as part of a tentative trap definition. For example, while testing a new *⎕TRAP*, you might include in each trap line a reference to *TRACE*, so that a trap definition might look like this:

```
evnums E TRACE ⎕ER ◇ FIXUP ◇
                          →RESUME
```

In that example, *FIXUP* and *→RESUME* are statements you expect to remain in the final version, but *TRACE* is included solely for testing. You can easily have *TRACE* report the circumstances in which the trap was invoked, or have it halt (with a local *⎕TRAP* of its own) so that you regain control.

3. Provide default trap definitions for events not otherwise covered in *⎕TRAP*. If you make *⎕TRAP* local to a function you're building, but fail to specify trap definitions for some events, you are in effect letting a more global *⎕TRAP* (if there is one) decide what will happen when one of those events occurs. Until you're really ready to do that, it would be prudent to append to your *⎕TRAP* a definition such as `0 1000 S` (meaning "Stop after any error or any interrupt"). That will assure a halt for any event you haven't anticipated.

4. When testing a function with a local *⎕TRAP* establish a default *⎕TRAP* outside it. When a trappable event occurs during execution of a function with a local *⎕TRAP*, the system first examines the local *⎕TRAP*, and, if that doesn't cover the event, looks next at the global *⎕TRAP* in effect when the function was invoked. Until your function is behaving satisfactorily, make sure that the global *⎕TRAP* will handle unexpected events. The simplest way to do that is to make the global *⎕TRAP*'s first trap definition something such as `0 1000 S`.

5. Use *⎕SIGNAL* to test error conditions that may not otherwise arise conveniently. You may have included a trap for an event that occurs only rarely and is inconvenient to arrange. You can test the working of the trap by using *⎕SIGNAL* to generate a report of the error you wish to trap. Because *⎕SIGNAL* aborts execution of the function that used it, it is usually preferable to embed it in a defined function such as *ERRSIGNAL*:

```
    ∇ ERRSIGNAL N;⎕TRAP
[1]   ⎕TRAP←'0 1000 S'
[2]   ⎕SIGNAL N
    ∇
```

You could then invoke the test conditionally, by inserting a reference to it at the start of line where you anticpate an error:

```
MAINFN[..]   ERRSIGNAL C/evnums ◇ WORK
```

In the foregoing, **evnums** are the events to be trapped and *C* the condition under which they are to be trapped; *WORK* represents the normal contents of the line.

6. Consider a trap definition that resets *⎕TRAP*. A trap definition may include in a trial version of its trap an instruction which resets *⎕TRAP* to something that will prevent the original *⎕TRAP* from being inadvertently invoked again. For example, a trap definition might be as follows:

```
evnums E ⎕TRAP←'∇0 1000 S' ◇
                    FIXUP ◇ →RESUME
```

After the first use of this *⎕TRAP*, a subsequent error will encounter the new value of *⎕TRAP*, and work will stop.

7. Disable *□TRAP* in the workspace saved at halt of an N-task or B-task. When you design a workspace to be run in batch mode, there are several precautions to be observed. You want the latent expression to start the task correctly, but you want to avoid having the task start automatically at a terminal.

Whenever a batch task halts, unless its active workspace was cleared, the system saves it under the **savews** name provided in the instruction that started it. If the task halted because of an error that wasn't successfully trapped, you'll want to load **savews** to find out what went wrong. Exploring a workspace in which *□TRAP* has been set can cause you some surprises; any error during your explorations may be trapped and may cause execution to be resumed. To avoid that possibility, it's good practice to set up the latent expression so that when you subsequently load **savews**, **(a)** execution does **not** automatically resume, and **(b)** the visible value of *□TRAP* is saved (so thatt you can examine it) and immediately reset it to `0 1000 S`.

The second precaution is included to assure that your efforts to examine the contents of the workspace are not trapped, and do not cause execution to be resumed.

Other points regarding design of the latent expression for a batch task are discussed in the manual "Batch Tasks in SHARP APL."

Special precautions when trapping interrupts: Writing a trap definition for interrupt events requires a great deal of caution.

When a function behaves improperly, it's important to be able to interrupt it. Whenever you set a trap for an interrupt, you are defining an alternate way for the system to respond to that interrupt, in place of its usual behaviour. You are thereby disabling (even if only temporarily) some of the built-in safety features of the system. For example, if you set up a trap definition which tells the system to resume execution following "attention" or "interrupt," as long as that trap definition is in effect, you have forfeited all ability to halt work by the usual signals you can send from the terminal (pressing ATTN or BREAK).

When you set up a trap definition that intercepts an "input interrupt" (that is, *0* BACKSPACE *U* BACKSPACE *T*) you may find yourself with no means to escape from a request for character input. And if the program is in a loop, there may be nothing at all you can do to regain control.

Warning: It is quite possible to write a defective *□TRAP* that will make your program run endlessly, immune to most of the normal ways of halting it. That can be both annoying and expensive. If you make that mistake, the only recourse is to have your task bounced, or, if it's a T-task, to disconnect the terminal.

Responsibility to avoid prolonged unresponsiveness: When you're using a high-performance interactive system such as SHARP APL, you have a right to expect that the system will execute your work promptly, and will respond promptly to emergency signals such as "attention" or "interrupt." So do other people who may use programs you've written. There are situations in which it is clearly desirable to trap interrupts. Those are situations in which, either for security or for convenience, you don't want the user to be able to interrupt a particular stage of the work. As author, you have a particular responsibility not to write programs that become unresponsive.

Whenever the system seems no longer to heed signals from the keyboard, any user should assume that something is wrong. It is normal to suspect that there has been a failure of the system itself or of the communication link between it and your terminal. If you write a program that causes the terminal to become unresponsive for a significant interval—more than a second or two—you run the serious risk that you (or others who use your program) will be unable to distinguish that unresponsiveness from a system failure.

DC-10 Operating Cost Data for 1978

1978 *OPERATING AND COST DATA:DC-10*

	AA	*UA*	*CO*
OPERATING EXPENSES			
(DOLLARS PER BLOCK HOUR)			
FLYING OPERATIONS:			
CREW EXPENSE	432	441	419
FUEL,OIL ϵ TAXES	892	839	893
INSURANCE	13	12	21
OTHER EXPENSES			
TOTAL	1,337	1,292	1,333
DIRECT MAINTENANCE:			
AIRFRAMES	123	98	102
ENGINES	241	230	253
OTHER FLIGHT EQUIP.	14		26
TOTAL	378	328	382
RENTALS			
FLIGHT DEPREC.	277	323	325
TOTAL WRITE-OFF	278	323	325
TOTAL DIRECT EXPENSE	1,993	1,944	2,040
MAINT. BURDEN	209	198	124
TOTAL AIRCRAFT EXPENSE	2,203	2,141	2,164
AVAIL. SEAT MILES	10,618,844	14,042,415	6,461,153
DIRECT COST PER ASM	0.01504	0.01518	0.01656
DIRECT COST PER PLANE MILE	3.97	3.83	3.92
AVERAGE HOP:			
LENGTH (REV S.M.)	1,438	1,327	1,330
DURATION (REV AIRBORNE)	2.92	2.76	2.72
TOTAL BLOCK HOURS	93,100	131,549	62,401
FUEL PRICE PER GALLON	0.390	0.381	0.383
FUEL COST PER BLOCK HOUR	861	812	859
FUEL BURN PER BLOCK HOUR	2,210	2,130	2,243

HOW FUZZY IS FUZZ?

Gene McDonnell, Palo Alto

I find that I don't have a very good feeling for numbers with large exponents (positive or negative). In trying to develop such a feeling for fuzz, I came up with some analogies which helped me appreciate what was being discussed. Perhaps if you are in the same fix they will help you too.

As you know, fuzz is a relative quantity, and gets larger as the numbers involved get larger. What does it mean to say that relative fuzz for the number 1E9 is equal to ⌷CT×1E9? How about ⌷CT×1E5, or even, down at the single digit level, ⌷CT itself?

Suppose you were standing on the number line at the number 999,999,999 and were 100 meters from the next higher number, 1,000,000,000, and had to hit a target that was as big as relative fuzz for that number. How big is the target? About 2.84 mm in diameter. Is that good enough accuracy?

Now let's change scale and imagine that the number line is wrapped around the earth, and that we are standing in Wellington, New Zealand, where the number 99,999 is, and that we want to hit the number 100,000, at a target located 20,000 km, or halfway around the earth, away, at the antipode to Wellington, a spot halfway between Salamanca and Valladolid, in Spain. How big is that spot? It is 5.7 cm in diameter. That's right, centimeters.

Lastly, suppose now that we change scale again, and run the number line between the earth and the sun, and stand on the earth at zero on the line, and have the number one at the sun, 150,000,000 km away. Now how big is the target? It is just 4.26 mm in diameter. Millimeters!

Perhaps this analogy will reassure you about using fuzzed functions. This last picture tells us that for the number one and another number to be considered equal, the second number will have to be as close to it as is implied by a target 4.26 mm in diameter at a distance of 150,000,000 km.

That's close, indeed.

Shortly after the first of September the residue and encode functions will be fuzzed in SHARP APL — see pages T1-T3 for details.

PEOPLE and PLACES

OSLO

19 May, 1979: It was a clear but cool Norwegian day when David Bonyun and I arrived at Oslo Central Station — the beginning of a four month stay to establish a new subsidiary, I.P. Sharp A/S. This new office is the third Sharp office in Scandinavia, the other two being in Copenhagen and Stockholm. The first day in Oslo proved to be rather hectic as several potential customers were anticipating the arrival of the Sharp team.

The first morning we met with Carl Storm of Storm Systems. Over a breakfast of coffee and pastries a business agreement and a friendship were both established. This was the first indication that Sharp would have a role to play in the Norwegian data processing industry.

The work schedule was interrupted on May 17th when all of Norway celebrates independence day. The main road, Carl Johans Gate, which leads to the Royal Palace, was jammed with people despite the steady drizzle, some in traditional costume. Norwegian flags flew from poles lining the streets. Brass bands paraded up and down. Everyone was there. Deciding to take advantage of a quiet day at work, David dragged me with him to the office but, finding the main doors locked and the building deserted, we both joined in the celebration with the rest of Norway.

It is hard to judge a place after one short week but first impressions are important: we already wish our stay in Oslo were for more than four months.

Alistair McKinnell

SYDNEY:

On Opening a Sharp Office

Isaac Ehrlich

It's 1:35pm Friday the 2nd of March 1978, and Sydney Australia has finally linked up to the Toronto data centre. Three weeks later the alpha concentrator is installed —— SHARP APL is now a local call.

Arrival was on a hot, humid Friday evening in mid December 1977 — hottest summer temperatures in the last 20 years — 104 degrees Fahrenheit at 7:30pm when the plane landed! Good sunbathing and beach weather especially for the Christmas and summer vacation period down in the antipodes. The weekend was spent familiarizing myself with the city. Sydney is one of the prettiest harbour cities in the world, with people naturally drifting towards the sea, sailboats dot the horizon from the early morning.

Monday morning paperwork began. Three months of slugging at multipart forms finally resulted in our Australian company's incorporation in the major Australian States. Very shortly afterwards we received permission for our overseas telecommunications lines (undersea cable) and the okay for linking these to internal lines. Eight lines were ordered for local access dialing from Telecom Australia, the internal communications company. Coordination was fairly smooth, our lines both internal and external to Australia were tested. We're up!

The alpha was delivered to us in slightly damaged condition: it seems that FRAGILE on a cargo label means 'dump off the plane'. Our normally rectangular box arrived slightly skewed. A slight hammer job and we had a reasonably functioning alpha. For several weeks we ran with about one down line load per day. By the time the lines were hardwired at the overseas terminal and at Telecom offices we had a new alpha and our lines were steady. At this time our first customers started to use SHARP APL. One problem initially was the mid-afternoon shutdown of APL (midnight in Toronto) for backup purposes. This was remedied later in the year with the introduction of on-line backup.

The first few months saw many tests performed: the types of available terminals, the quality of lines, the availability of modems and the setting up of systems. Line drops were frequent, but the customers bore with us, so that within three months we saw Melbourne would also be a centre of increasing trade for the Sharp system. Lines were ordered. By the end of June these also were installed and our beachhead was finalized.

One year later and we see SHARP APL picking up at an ever increasing pace. IBM has announced that they are going out of the timesharing business in Australia. This hopefully will help to spread our name further since only two companies offer APL in Australia at the moment — Control Data and ourselves.

I would like to thank all those Australians who helped a foreigner to both set up and conduct business in Australia.

CALGARY:

We are pleased to introduce to you several new additions to the I.P. Sharp Calgary office:

Scott Remborg comes to us from Norway via Edmonton. Scott recently graduated from the University of Alberta with an MBA degree. He obtained his undergraduate degree from Oslo in Business Administration, which included a heavy emphasis on Management Science. With his business background and his interest in finance and the petroleum industry, we feel he will be a valuable addition to the Calgary office.

Linda Furrow graduated from the Massachusetts Institute of Technology with a BSc degree in Mathematics in 1970. For the past 5 years, Linda has lived in Calgary, where she has worked for a major Canadian oil company and later as an independent data processing consultant. Linda has worked with a variety of computer systems and computer languages, implementing systems ranging from a job accounting package to an on-board data acquisition system for Canada's cargo-carrying icebreaker. We are pleased to have Linda with us.

George Hart is a native Albertan who graduated from the University of Calgary with a BSc degree in Statistics in 1976. Prior to his joining Sharp, George worked as a research statistician for the Board of Education where he provided Statistical Analysis for research and evaluation projects. Welcome, George.

EDMONTON:

Jim Wesley has joined the Edmonton office in the customer support area. Jim is a Mathematics graduate of the University of Alberta.

APL CLASSROOM

INTRODUCTION TO APL

Amsterdam
(Dutch)
July 18-20

Atlanta
(5 day)
September 24-28

Birmingham
(3 day)
July 10-12
August 14-16
October 16-18
November 30-Dec. 2
December 11-13

Calgary
(5 day)
September 5,6,12,13,19

Dallas
(4 day)
July 10,12,17,19
August 7,9,14,16

London
(3 day)
July 25-27
September 10-12

Montreal
(French/English)
July 9-11/24-26

Ottawa
(non-programmers)
August 7-9
October 2-4
December 4-6

(programmers)
July 3-5
September 4-6
November 6-8

Rochester
July 23-27
August 20-24
September 24-28,
October 15-19
November 19-23

San Francisco
(4 half-days)
July 10-13
August 21-24
October 2-5
November 13-16

Toronto
(3 day)
July 16-18
August 6-8
September 4-6
September 24-26
October 15-17

SEMINARS

Birmingham
- Error Trapping, July 31
- Highspeed Print, June 29
- N & B-tasks, August 31
- Plotting, October 31
- Shared Variables, September 28
- Using Files, December 17

Gloucester
- Actuarial, August 20
- Financial, July 16
- Linear Programming, September 17
- MABRAUTIL, July 9
- Report Formatting, September 10
- Security, September 24

London
- N- and B-tasks, September 6

Montreal
- Financial Analysis, July 5-6

Ottawa
- Batch (Non-Terminal) Tasks, August 21, Dec. 18
- Box-Jenkins, October 23
- Highspeed Printing, July 24, November 22
- Input Validation, October 25
- MAGIC, September 20, December 20
- Plotting, August 22
- Regression Analysis, August 23, November 20

Rochester
- Design of Files & Data Structures, July 11
- Restartability & Reliability, July 12

San Francisco
- Graphics, July 18, Sept. 11, Nov. 6
- MAGIC, September 19

Toronto
- Actuarial APL Techniques, Aug. 13, Oct. 9, Dec. 10
- AIDS - Introduction, Sept. 17-18, Dec. 4-5
- Box-Jenkins, July 9, Sept. 7, Nov. 8
- Data Base Design, August 22, Oct. 19, Dec. 20
- Forecasting Methods, July 11, Sept. 13, Nov. 15
- MAGIC for Time Series Analysis, July 12, Sept. 14
- Plotting, July 19, Sept. 19, Nov. 16
- Regression Analysis, August 16, Oct. 18, Dec. 14
- Report Formatting, August 23, Oct. 25, Dec. 28
- Saving Money with N- and B-tasks, August 2, Oct. 4
- Text Editing, August 27-29, Nov. 19-21

SPECIAL COURSES

'Appreciation of APL'
- **Gloucester**, July 2, September 3
- **London**, July 31, September 20
- **Warrington**, September 19

'APL Review'
- **London**, July 2, August 8, September 24

'APL for Managers'
- **Toronto**, September 20, November 23

ADVANCED APL

'APL and System Design'
- **London**, July 19, September 3

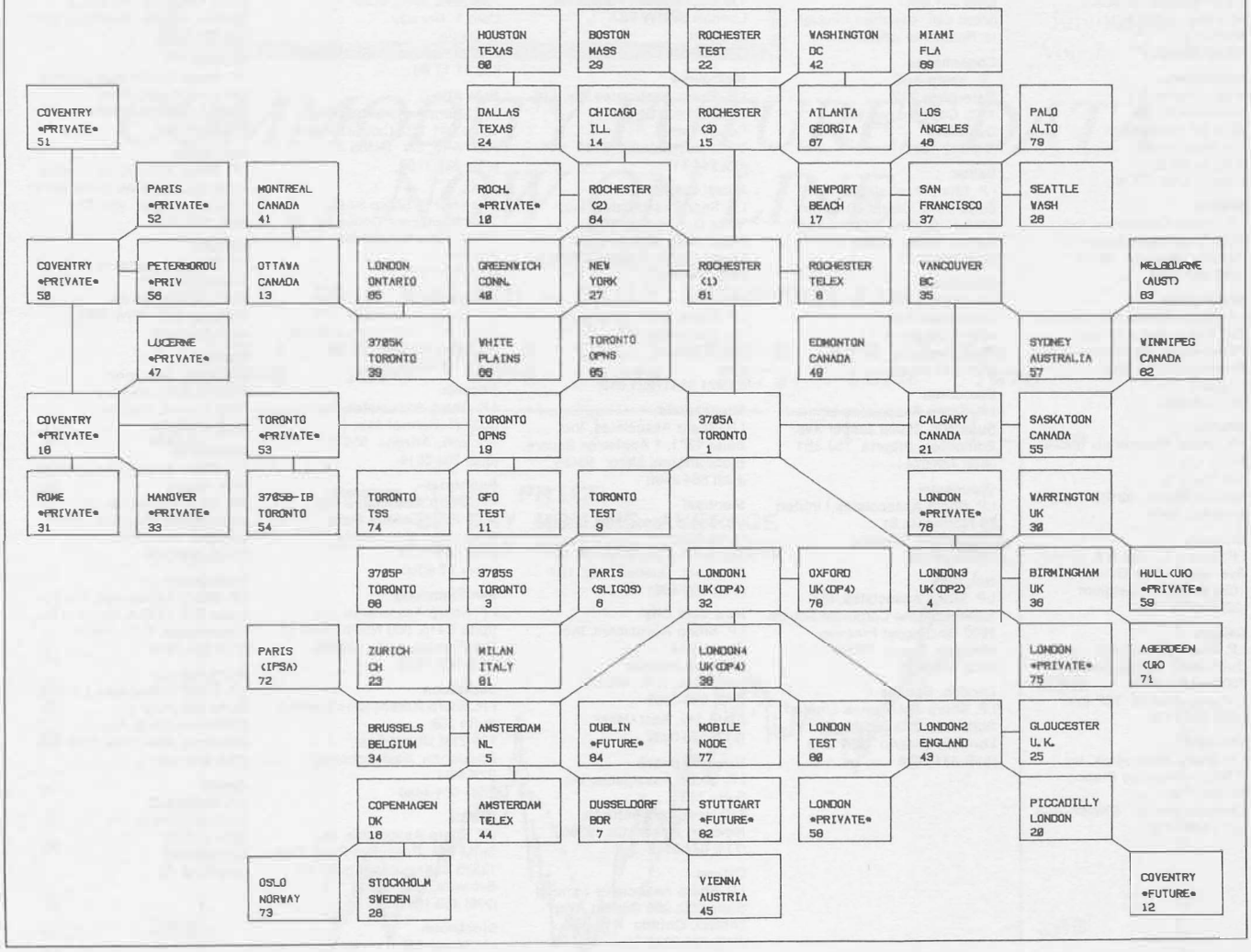
'Advanced APL'
- **San Francisco**, August 7

'Advanced APL and Efficient Coding Techniques'
- **Toronto**, August 20-21, November 1-2

INTERMEDIATE APL

- **Gloucester**, August 13
- **London**, July 3, August 9, September 25
- **Montreal**, July 16-18
- **Ottawa**, September 18,19
- **San Francisco**, October 9, December 4
- **Stockholm**, (in Swedish or English)
- **Toronto**, July 23-25, September 10-12, Nov. 12-14

Network Typology



UPDATE

☐ Please amend my mailing address as indicated.

☐ Add to your mailing list the following name(s).

☐ Send me a SHARP APL publications order form.

Name: _____

Co.: _____

Address: _____

I.P. Sharp Associates **Head Office:** 145 King Street West, Toronto, Canada M5H 1J8 (416) 364-5361

International Branch Offices

Aberdeen
I.P. Sharp Associates Limited
5 Bon Accord Crescent
Aberdeen AB 1 2DH
Scotland
(0224) 25298

Amsterdam
Intersystems B.V.
Herengracht 244
1016 BT Amsterdam
The Netherlands
(020) 24 40 50
Telex: 18795 ITS NL

Atlanta
I.P. Sharp Associates, Inc.
5000 Snapfinger Woods Dr.
Decatur, Georgia 30035
(404) 987-2301

Birmingham
I.P. Sharp Associates Limited
2nd Floor, Radio House
79/81 Aston Rd. North
Birmingham B6 4BX
England
021-359-6964

Boston
I.P. Sharp Associates, Inc.
Suite 415
148 State St.
Boston, Mass. 02109
(617) 523-2506

Brussels
I.P. Sharp Europe S.A.
Ave. General de Gaulle, 39
1050 Brussels, Belgium
(02) 649 99 77

Calgary
I.P. Sharp Associates Limited
Suite 2660, Scotia Centre
700-2nd St. S.W.
Calgary, Alberta T2P 2W2
(403) 265-7730

Chicago
I.P. Sharp Associates, Inc.
2 North Riverside Plaza
Room 1746
Chicago, Illinois 60606
(312) 648-1730

Cleveland
I.P. Sharp Associates, Inc.
(216) 431-6861
(local call, switched through to Rochester office.)

Copenhagen
I.P. Sharp ApS
Østergade 24B
1100 Copenhagen K
Denmark
(01) 112 434

Dallas
I.P. Sharp Associates, Inc.
Suite 1148, Campbell Centre
8350 N. Central Expressway
Dallas, Texas 75206
(214) 369-1131

Düsseldorf
I.P. Sharp GmbH
Leostrasse 62A
4000 Düsseldorf 11
West Germany
(0211) 57 50 16

Edmonton
I.P. Sharp Associates Limited
Suite 505, 10065 Jasper Ave.
Edmonton, Alberta T5J 3B1
(403) 428-6744

Gloucester
I.P. Sharp Asssociates Limited
29 Northgate St.
Gloucester, England
(0452) 28106

Houston
I.P. Sharp Associates, Inc.
Suite 925, One Corporate Square
2600 Southwest Freeway
Houston, Texas 77098
(713) 526-5275

London, Canada
I.P. Sharp Associates Limited
Suite 510, 220 Dundas St.
London, Ontario N6A 1H3
(519) 434-2426

London, England
I.P. Sharp Associates Limited
132 Buckingham Palace Rd.
London SW1W 9SA
England
(01) 730-0361

Melbourne
I.P. Sharp Associates Pty. Ltd.
36 Elizabeth Street
South Yarra
Victoria, Australia 3141
(03) 244-417

Miami Lakes
I.P. Sharp Associates, Inc.
Suite D, Kennedy Bldg.
14560 N.W. 60th Avenue
Miami Lakes, Florida 33014
(305) 556-0577

Milan
I.P. Sharp Srl
Via Eustacchi, II
20129 Milan
Italy
(2) 271 6541/221 612

Minneapolis
I.P. Sharp Associates, Inc.
Suite 1371, 1 Appletree Square
Bloomington, Minn. 55420
(612) 854-3405

Montreal
I.P. Sharp Associates Limited
Suite 1610
555 Dorchester Blvd. W.
Montreal, Quebec H2Z 1B1
(514) 866-4981

New York City
I.P. Sharp Associates, Inc.
Suite 2004
200 Park Avenue
New York, N.Y. 10017
(212) 986-3366
Suite 242, East. Mezz.
(212) 599-0232

Newport Beach
I.P. Sharp Associates, Inc.
Suite 1135
610 Newport Centre Dr.
Newport Beach, Ca. 92660
(714) 644-5112

Ottawa
I.P. Sharp Associates Limited
Suite 600, 265 Carling Ave.
Ottawa, Ontario K1S 2E1
(613) 236-9942

Oslo
I.P. Sharp A/S
Postboks 1470, Vika
Oslo 1, Norway
4. Etasje
Krønprinsesse Märthas Plass 1
(02) 41 17 88

Palo Alto
I.P. Sharp Associates, Inc.
Suite 201, 220 California Ave.
Palo Alto, Ca. 94306
(415) 327-1700

Paris
Société I.P. Sharp SARL
Tour Neptune - Cèdex No. 20
92086 Paris-la-defense
France
(1) 773 57 77

Philadelphia
I.P. Sharp Associates, Inc.
Suite 407, 1420 Walnut Street
Philadelphia, PA. 19102
(215) 735-3327

Phoenix
I.P. Sharp Associates, Inc.
3033 N. Central Ave.
Phoenix, Arizona 85012
(602) 264-6819

Rochester
I.P. Sharp Associates, Inc.
1200 First Federal Plaza
Rochester, N.Y. 14614
(716) 546-7270
Telex 97-8380

San Francisco
I.P. Sharp Associates, Inc.
Suite C415, 900 North Point St.
San Francisco, Ca. 94109
(415) 673-4930

Saskatoon
I.P. Sharp Asssociates Limited
Suite 208
135 21st Street East
Saskatoon, Saskatchewan
S7K 0B4
(306) 664-4480

Seattle
I.P. Sharp Associates, Inc.
Suite 217, Executive Plaza East
12835 Bellevue-Redmond Rd.
Bellevue, Wa. 98005
(206) 453-1661

Stockholm
I.P. Sharp AB
Kungsgatan 65
S111 22 Stockholm, Sweden
(08) 21 10 19

Sydney
I.P. Sharp Associates Pty. Ltd.
Suite 1342, 175 Pitt Street
Sydney, N.S.W., Australia 2000
(02) 232-5914

Toronto
I.P. Sharp Associates Limited
145 King Street West
Toronto, Ontario M5H 1J8
(416) 364-5361

Vancouver
I.P. Sharp Associates Limited
Suite 604, 1112 West Pender St.
Vancouver, B.C. V6E 2S1
(604) 682-7158

Victoria
I.P. Sharp Associates Ltd.
Chancery Court
1218 Langley Street
Victoria, B.C. V8W 1W2
(604) 388-6365

Vienna
I.P. Sharp Ges. mbH
Rechte Wienzeile 5/3
1040 Vienna, Austria
(222) 57 65 71

Warrington
I.P. Sharp Associates Limited
Paul House
89 - 91 Buttermarket St.
Warrington, Cheshire
England WA1 2NL
(0925) 50413/4

Washington
I.P. Sharp Associates, Inc.
Suite 307, 1730 K Street N.W.
Washington, D.C. 20006
(202) 293-2915

Winnipeg
I.P. Sharp Associates Limited
Suite 208 (Aug. 1)
213 Notre Dame Ave.
Winnipeg, Manitoba R3B 1N3
(204) 947-1241

Zurich
I.P. Sharp A.G.
Badenerstrasse 141
8004 Zurich
Switzerland
(1) 241 52 42

SHARP APL Communications Network: Local Access Cities

APL OPERATOR VOICE (416) 363-2051 COMMUNICATIONS (416) 363-1832

Local dial access is available in all locations listed above. The SHARP APL Communications Network also provides local dial access in:

• Ann Arbor • Buffalo • Coventry • Dayton • Des Moines • Detroit • Ft. Lauderdale • Greene (NY) • Greenwich (Ct)
• Halifax • Hamilton • Kitchener • Liverpool • Los Angeles • Manchester • Raleigh • Regina • Syracuse • White Plains (NY).

In the United States the SHARP APL Network is interconnected with the networks of TYMNET and TELENET to provide local dial access in more than 100 other cities.

The Newsletter is a regular publication of I.P. Sharp Associates. Contributions and comments are welcome and should be addressed to: Jeanne Gershater, I.P. Sharp Newsletter, 145 King Street West, Toronto, Canada M5H 1J8.
Jeanne Gershater, *Editor*
Ginger Kahn, *Assistant Editor*