INHALT

Impressum

ISSN 1438-4531

Herausgeber: Rainer Nußbaum, APL-Germany e.V., Mannheim

Redaktion: Dipl.-Volksw. Martin Barghoorn (verantwortlich)
Franklinstr. 28
D-10587 Berlin
Tel. (030) 314 24392
Fax (030) 314 25901
eMail: barg@cs.tu-berlin.de

Erscheinungsweise: halbjährlich

Erscheinungsort: Berlin

Verlag: RHOMBOS-VERLAG, Berlin
Kurfürstenstr. 17
D-10785 Berlin
Tel. (030) 261 9461
Fax (030) 261 6300
eMail: verlag@rhombos.de,
Internet: www.rhombos.de

Gestaltung: RHOMBOS-VERLAG, Berlin

Druck: dbusiness.de GmbH, Berlin

Homepage des APL-Germany e.V.: http://www.apl-germany.de
Homepage des APL-Journals beim Verlag: http://www.rhombos.de/shop/a/show/type/?2

Wie immer freuen wir uns auf Ihr Feedback zu diesem Heft. Schreiben Sie uns an barg@cs.tu-berlin.de, oder faxen Sie an: 030 - 314 25901.

Technische Hinweise für Autorinnen und Autoren

Manuskripte werden von der Redaktion entgegengenommen. Sie müssen im Besitz der Nutzungsrechte für die von Ihnen eingereichten Texte, Fotos und Zeichnungen sein. Die Zustimmung zum Abdruck wird vorausgesetzt.

Manuskripte und Beiträge aller Art werden in jeder Form entgegengenommen. Es werden jedoch Beiträge in gespeicherter Form bevorzugt. Bitte speichern Sie Ihren Text im Format Ihrer Textverarbeitung und zusätzlich als RTF-Datei (Rich Text Format). Sie brauchen sich nicht die Mühe machen, Ihren Text aufwendig zu formatieren. Auszeichnungen im Text sollten sich auf „Kursiv“ beschränken. Ihre Datei schicken Sie der Redaktion bitte auf einem Datenträger (Diskette oder CD-ROM). Alternativ hierzu können Sie der Redaktion Ihre Daten auch per Email (barg@cs.tu-berlin.de) schicken.

Wenn Sie Ihr Manuskript einreichen, sollten Sie zusätzlich eine Gliederung und Zusammenfassung Ihrer Arbeit vorlegen.

Wichtig: In der Zusammenfassung (abstract) sollte in wenigen Sätzen die wichtigste Aussage Ihres Textes wiedergegeben sein.

Grafiken

Bitte senden Sie uns einen sauberen Ausdruck der Graphiken auf rein-weißem Papier. Linien (keine Haarlinien!) und Schriften in den Graphiken sollten so groß gewählt sein, dass sie auch bei (stark) verkleinerter Wiedergabe im Heft noch zu erkennen sind; auf der Rückseite vermerken Sie bitte Ihren Namen und die Nummer der Abbildung; Zusätzlich wäre es schön, wenn Sie uns die mit einem gängigen Vektor-Grafikprogramm erstellten Zeichnungen und Grafiken auch als Datei liefern. Grafikdateien (Strichzeichnungen) sollten möglichst im Austauschformat EPS (skalierbar) mit 300 bis 600 dpi Auflösung in schwarz-weiß bzw. Graustufe abgespeichert werden. Nach Rücksprache können sie zusätzlich Ihre Grafiken in Ihrem Original-Dateiformat liefern oder in ihr Textdokument einbetten.

Digitale Fotos benötigen wir im TIF-Format mit einer maximalen Auflösung von 250 dpi. Nicht geeignet sind Abbildungen und Fotos, wenn Sie aus dem Internet stammen beziehungsweise im Original als Bildschirmdarstellung (Auflösung 75 - 150 dpi) vorliegen. Ebenfalls nicht geeignet sind Grafiken, die mit einem pixelorientierten Zeichenprogramm erstellt wurden.

Liebe APL Freundinnen und APL Freunde,

bei unserer zweiten Tagung in 2005 sind wir wie vom Wetterbericht angekündigt, heftig eingeschneit worden. Glücklicherweise hatten wir bei der Thomas Cook AG einen sehr freundlichen Gastgeber für unsere Tagung gefunden. Besonders bedanke ich mich bei Herrn Peter Odenthal von Thomas Cook, der für uns die hervorragende Organisation der Tagungsräumlichkeiten übernommen hat. Neben Vorträgen von APL Herstellern und APL Anwendern wurde uns auch ein Einblick in APL Anwendungen bei Thomas Cook gewährt.

An dieser Stelle sei auch den Referenten für ihre Arbeit gedankt. Gleichzeitig möchte ich wieder die Bitte an potentielle Referenten richten, uns weiterhin interessante Vorträge zu liefern.

Als Vorabinformation möchte ich bekannt geben, dass unsere Herbsttagung im Jahre 2006, welche für den 27./28. Novemer geplant ist, unter dem Motto „40 Jahre IBM APL“ steht. Auch diese Tagung wird sicher wieder interessant werden, so dass Sie sich den Termin für einen Besuch der Tagung reservieren sollten. Nähere Datails werden wir rechtzeitig bekannt geben. Bitte besuchen Sie dazu auch unsere web-site apl-germany.de.

Reiner Nussbaum

EDITORIAL

Dr.Reiner Nussbaum

Ein Array Processing File System

gift-ALU APFS: das Dateisystem lernt rechnen

Zusammenfassung

Idee des Array Processing File Systems ist das Bereitstellen mathematischer Funktionen im Dateisystem. Als Beispielanwendungen behandelt dieser Beitrag das Lösen quadratischer Gleichungen, das Invertieren von Matrizen, das Multiplizieren von Matrizen sowie das Verarbeiten von Produktionsmatrizen. Die zu verarbeitenden Daten werden entweder direkt als Dateien bereitgestellt oder zumindest die Information, wie auf die Verarbeitungsdaten zugegriffen werden kann (z.B.: SQL-Befehle, wenn sich die Daten in der Datenbank befinden). Die Verarbeitung wird gestartet, in dem eine Triggerdatei in ein bestimmtes Verzeichnis gestellt wird.

Ein Hintergrundprozess untersucht fortlaufend das Dateisystem nach solchen Triggerdateien. Wenn eine Datei gefunden wird, so kann anhand des Verzeichnisses, in dem sich die Datei befindet, unmittelbar die Verarbeitung stattfinden. Die Art der Verarbeitung sowie alle weiteren Verarbeitungsparameter können aus dem Ort abgeleitet werden, an dem sich die Triggerdatei befindet.

Soweit ein standardisiertes Anwendungsumfeld für die mathematischen Verfahren benutzt wird, welches aus dem Ort der Triggerdatei abgeleitet wird, sind die eingesetzten Verfahren in der Regel unmittelbar für eine parallele Verarbeitung geeignet.

Bei der Nutzung von shared file systems ist es mit diesem Ansatz möglich, dass der mathematische Teil auf einer / mehreren anderen Maschinen erfolgt (Cross System Processing) oder sogar auf anderen Rechnerplattformen, wie Unix, Linux usw. (Cross Platform Processing). So wäre es beispielsweise auch möglich, komplizierte mathematische Funktionen, welche in APL oder C geschrieben sind, auch von Cobol Programmen aus zu benutzen.

1. Grundidee

Die Idee des Array Processing File Systems ist, dass in Verzeichnissen numerische Informationen in der Form von Arrays, d.h. Vektore und Matrizen bereitgestellt werden. Nachdem alle für die Verarbeitung erforderlichen Arrays vorliegen, wird diese veranlasst, in dem eine (leere) Triggerdatei mit einem verabredeten Namen in das Verzeichnis gestellt wird. Nach der geforderten Verarbeitung werden die Ergebnisse in dem gleichen Verzeichnis zurückgegeben.

Das Bereitstellen der Informationen kann etwa in menschenlesbarer Form als Textdateien, oder in einer internen Darstellung erfolgen. Alternativ dazu ist es auch denkbar, dass etwa in einer Datenbank entsprechende Tabellen abgespeichert werden oder SQL Befehle hinterlegt werden, anhand derer die numerischen Daten (Arrays) ermittelt werden können.

Mit einer solchen Vorgehensweise können Verarbeitungsabläufe so gestaltet werden, dass beispielsweise die Datenbeschaffung bzw. Datenpräsentation von einem Programmteil erfolgt, während der numerische Teil von einem davon unabhängigen Programm abgearbeitet wird. Die Trennung hat darüber hinaus den Vorteil, dass die numerischen Funktionen damit auch für Programmiersprachen zugänglich gemacht wird, welche für mathematische Probleme eigentlich nicht gedacht sind, etwa COBOL. Genauso ist es an dieser Stelle möglich, die Verarbeitungsteile auf verschiedene Rechner mit ggf. unterschiedlichen Plattformen zu verteilen.

2. (Beispiel-)Funktionen

Die Grundidee wird nun anhand von Beispielfunktionen verdeutlicht. Als Basis werden einige einfache mathematische Probleme herangezogen. Zur Vereinfachung wird in allen Beispielen davon ausgegangen, dass die numerischen Werte als menschenlesbare Dateien vorliegen sollen. Damit ist es leicht möglich, ein standardisiertes Umfeld für die mathematischen Operationen zu beschreiben. Es sind vier Hauptbereiche für das Verarbeitungsumfeld erkennbar:

- Auswerten des Verarbeitungsumfeldes: wo befinden sich die Eingabewerte, was ist zu tun und wohin sollen die Ausgabewerte abgelegt werden?

- Beschaffen der Eingabewerte: die zu verarbeitenden Daten müssen für die Verarbeitung bereitgestellt werden; in dem hier betrachteten Fall ist dies das Einlesen der Text Arrays. Falls die Daten aus anderen Quellen, etwa einer SQL Datenbank beschafft werden müssen, so ist an dieser Stelle eine geeignete Anpassung durchzuführen.
- Numerische Verarbeitung der Daten: Nachdem die Daten bereitstehen und bekannt ist, was mit den Daten zu geschehen hat, kann in diesem Dritten Schritt die Verarbeitung stattfinden.
- Bereitstellen des Ergebnisses: In einem letzten Schritt wird das Ergebnis der numerischen Verarbeitung ausgegeben. Die dazu notwendigen Informationen wurden in dem ersten Schritt bei der Auswertung des Verarbeitungsumfeldes ermittelt. Wenn die Ergebnisdaten nicht wie hier angenommen als Textdaten auszugeben sind, so ist an dieser Stelle eine geeignete Modifikation durchzuführen.

Mit dieser allgemeinen Vorgehensweise können beliebige mathematische Aufgaben gelöst werden, die einen oder mehrere ‚Arrays' als Eingabe benutzen und als Ergebnis einen ‚Array' liefern. Da unterschiedliche Prozesse über die Verzeichnisstruktur getrennt werden, ist es sehr leicht möglich, eine Parallelverarbeitung für den numerischen Teil zu implementieren. Wie weiter unten beschrieben wird, ist auch die Verteilung auf verschiedene Rechner, ggf. auf unterschiedlichen Plattformen, leicht realisierbar.

Beispielhaft werden nun einige Lösungen beschrieben.

Lösen von quadratischen Gleichungen

In diesem Abschnitt wird das Verarbeiten eines einfachen Vektors, die Koeffizienten einer Quadratischen Gleichung, mit einem kurzen Programm gezeigt werden. Bei dem Lösen quadratischer Gleichungen steht man vor folgendem mathematisches Problem:

Allgemeine Form: (nur reelle Werte)

ax^2 + bx + c = 0

Koeffizienten a, b, c (a <> 0)

Lösung:

$$x1,2 = \frac{-b \pm sqr(2b-4ac)}{2a}$$

Diskriminante D = 2b-4ac

D > 0 - zwei Lösungen
D = 0 - eine Lösung
D < 0 - keine reelle Lösung

Das Umsetzen des Problems für ein array processing file system erfolgt in der nebenstehend beschriebenen allgemeinen Vorgehensweise. In der Funktion ALU_Q_GLEICHUNG wird zunächst das allgemeine Umfeld für die Verarbeitung ermittelt (ALU3_INIT). Der anwendungsbezogene Teil,

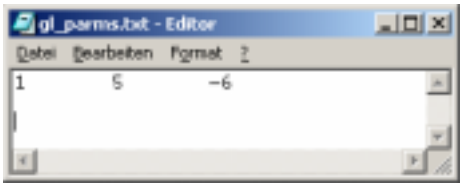
- das Einlesen der Daten aus der Datei „gl_parms.txt",
- die Verarbeitung und
- das Ergebnis in der Datei „A-Q_GLEICHUNG.TXT" wird bereit gestellt in der Funktion ALUAPPL_Q_GLEICHUNG. (siehe nachstehende Darstellung):

```
      VALU_Q_GLEICHUNG[⎕]∇
[0]   ALU_Q_GLEICHUNG
[1]    RC←ALU3_INIT
[2]    RC←ALUAPPL_Q_GLEICHUNG
    ∇
      VALUAPPL_Q_GLEICHUNG[⍞]∇ (Auszug)
[0]   RC←ALUAPPL_Q_GLEICHUNG;M;R;G_ERROR
[5]    M←READMAT 'gl_parms.txt'  ⍝ READ FROM FILE
[9]    R←Q_GLEICHUNG,M        ⍝ CALCUTALE SOLUTION
[10]   'A-Q_GLEICHUNG.TXT' ALU∆WRITEMAT⍕3 1⍴R
[13]   →0
    ∇
```

Das Lösen der Aufgabe „Quadratische Gleichung", geschieht in der Funktion Q_GLEICHUNG. (siehe nachstehende Abbildung). Dabei beschränken wir uns hier auf den Zahlenraum der reellen Zahlen.

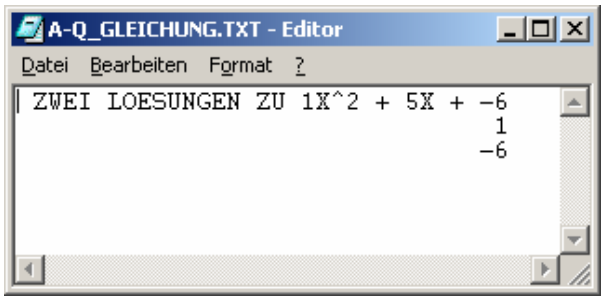
```
      VQ_GLEICHUNG[⎕]∇
[0]   O←Q_GLEICHUNG I;A;B;C;X;Y;Z;XX;TXT
[1]   ⍝ QUADRATISCHE GLEICHUNG LOESEN
[2]    ⎕IO←1
[3]    A←I[1]⋄B←I[2]⋄C←I[3]
[4]    O←3⍴⊂''
[5]    →(0>X←(B*2)-4×A×C)/NULL
[6]    →(0=X)/EINS
[7]   ZWEI:O[1]←⊂'ZWEI LOESUNGEN'
[8]    O[2]←((Y←-B)+Z←X*0.5)÷XX←2×A
[9]    O[3]←(Y-Z)÷XX
[10]   →0
[11]  NULL:O[1]←⊂'KEINE REELLE LOESUNG',TXT
[12]   →0
[13]  EINS:O[1]←⊂'EINE LOESUNG',TXT
[14]   O[2]←-B÷(2×A)
[15]   →0
      ∇
```

Ein Beispielablauf für das Lösen quadratischer Gleichungen ist folgender: Gegeben sind in der Datei gl_parms.txt die Werte:



Da für diese Problemstellung schon alle Eingabeinformationen vorhanden sind, kann die Verarbeitung veranlasst werden, in dem (für dieses Beispiel) eine leere Datei mit dem Namen doit.txt in das Verzeichnis gestellt wird. Natürlich könnte auch ein anderer Dateiname als doit.txt verabredet werden.

Das Ergebnis der Verarbeitung mit obigem Programm Q_GLEICHUNG ist:



Invertieren von Matrizen

Hier wird das Verarbeiten eines zweidimensionalen Arrays am Beispiel des Invertierens einer Matrix dargestellt. Das Problem stellt sich mathematisch wie folgt:

Invertieren von Matritzen

Allgemeine Form:

A * Inv(A) = Identity

Arbeitsweise:

Bereitstellen der Eingabewerte in der Datei „matrix.txt“.

Das Ergebnis wird geliefert in der Datei „A-INVMAT.TXT“.

Falls die Matrix nicht invertierbar ist, wird ein Hinweis in der Datei „ALUMSG.TXT“ ausgegeben.

Die Arbeitsweise in dem array processing file system ist im wesentlichen gleich. In der Funktion ALU_MATINVERT wird zunächst das allgemeine Umfeld für die Verarbeitung ermittelt (ALU3_INIT). Der anwendungsbezogene Teil,

- das Einlesen der Daten aus der Datei „matrix.txt“,
- die Verarbeitung und
- das Bereitstellen des Ergebnisses in der Datei „A-INVMAT.TXT“

geschieht in der Funktion ALUAPPL_MATINVERT.

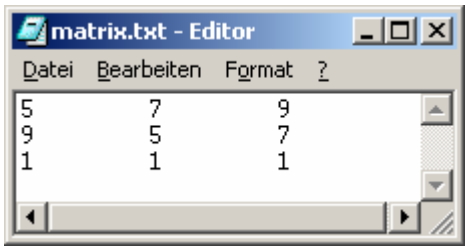
```
∇ALU_MATINVERT[⎕]∇
[0]    ALU_MATINVERT
[1]      RC←ALU3_INIT
[2]      RC←ALUAPPL_MATINVERT
     ∇
```

Bei der Lösung des Problems mit APL ist besonders interessant, dass das Invertieren der Matrix unmittelbar mit einer eingebauten Funktion lösbar ist (Zeile 8).

```
 ∇ALUAPPL_MATINVERT[⎕]∇
[0]    RC←ALUAPPL_MATINVERT;M;INV;R;G_ERROR
[1]    G_ERROR←RC←0                    ⍝ INIT GLOBAL ERROR SIGNAL
...
[5]    M←READMAT ⍞matrix.txt⍞          ⍝ READ FROM FILE
[6]    →(0≠G_ERROR)/ER01               ⍝ I/O ERROR
[7]    →(~=/⍴M)/ER02                   ⍝ SQARE MATRIX
[8]    ⍞→ER03⍞ ⎕EA ⍞INV←⌹M⍞           ⍝ PRODUCE MATRIX INVERSION
[9]    R←(GT∆IDENT 1↑⍴M)-INV+.×M       ⍝ CHECK FOR IDENTITY MATRIX
[10]   →(∨/1E¯10<,R)/ER03              ⍝ NON SINGULAR MATRIX
[11]   ⍞A-INVMAT.TXT⍞ ALU∆WRITEMAT INV
[12]   →(0≠G_ERROR)/ER04               ⍝ PROBLEMS WRITING INVERTED
                                         MATRIX
[13]   HOST ALU∆SYSPATH,G_DIRSEP,⍕alu3-sigok.bat⍕        ⍝ TELL RESULT
[14]   →0
[15]   ⍝ *** MESSAGES AND CODES ***
 ...
 [26]   ⋆0
     ∇
```

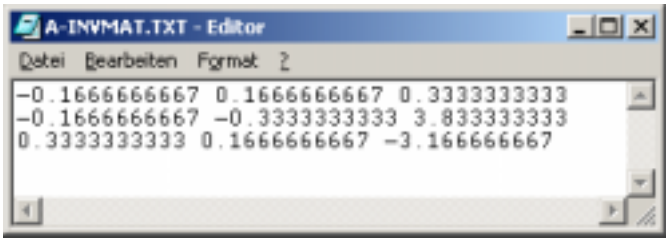
Ein Beispielablauf für das invertieren einer Matrix ist folgender:

Gegeben sind in der Datei matrix.txt die Werte



Da für diese Problemstellung schon alle Eingabeinformationen vorhanden sind, kann die Verarbeitung veranlasst werden, in dem – für dieses Beispiel – wieder eine leere Datei mit dem Namen doit.txt in das Verzeichnis gestellt wird. Natürlich könnte auch ein anderer Dateiname als doit.txt verabredet werden.

Das Ergebnis der Verarbeitung mit dem Programm ALUAPPL_MATINVERT ist



Multiplizieren von Matrizen

In diesem Abschnitt wird das Verarbeiten von zwei Arrays behandelt. Da die beiden Arrays in einer beliebigen Reihenfolge bereitgestellt werden können, wird hier deutlich, dass es sinnvoll ist, die Verarbeitung erst dann zu beginnen, wenn eine Triggerdatei vorliegt. Als Beispiel wird das multiplizieren von Matrizen verwendet. Das Problem ist:

Produkt zweier Matritzen

Allgemeine Form:

M = M1 * M2

Arbeitsweise:

- ➡ Bereitstellen der Eingabewerte in den Dateien „matrix-1.txt“ und „matrix-2.txt“.
- ➡ Das Ergebnis wird in der Datei „MATMULT.TXT“ bereitgestellt.
- ➡ Die Anzahl der Spalten von „matrix-1.txt“ muss mit der Anzahl der Zeilen in „matrix-2.txt“ übereinstimmen.
- ➡ Im Falle von Fehlern wird ein Hinweis in der Datei „ALUMSG.TXT“ ausgegeben.

Die Arbeitsweise in dem array processing file system beinhaltet nun das Verarbeiten zweier Arrays (Matrizen). Da diese in einer beliebigen Reihenfolge gespeichert werden können, wird an dieser Stelle deutlich, dass es eine wesentliche Vereinfachung ist, den Start der Verarbeitung an das Vorhandensein einer Triggerdatei zu binden. In der Funktion ALU_MATMULT wird wieder zunächst das allgemeine Umfeld für die Verarbeitung ermittelt (ALU3_INIT). Der anwendungsbezogene Teil,

- das Einlesen der Daten aus den Dateien „matrix1.txt“ und „matrix2.txt“,
- die Verarbeitung und
- das Bereitstellen des Ergebnisses in der Datei „MATMULT.TXT“

geschieht in der Funktion ALUAPPL_MATMULT.

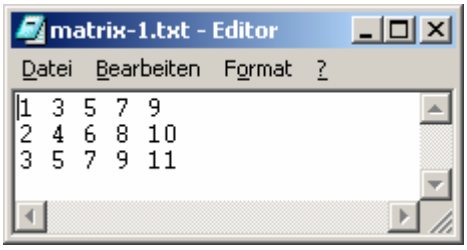
```
      ∇ALU_MATMULT[⎕]∇

[0]    ALU_MATMULT

[1]      RC←ALU3_INIT
[2]      RC←ALUAPPL_MATMULT
∇
```

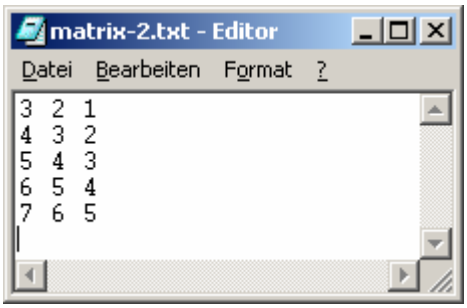
Bei der Lösung des Problems mit APL ist besonders interessant, dass die Matrizenmultiplikation unmittelbar mit einer eingebauten Funktion lösbar ist (Zeile 10).

```
∇ALUAPPL_MATMULT[⎕]∇
[0]    RC←ALUAPPL_MATMULT;M1;M2;RES;R;G_ERROR
[1]     G_ERROR←RC←0              ⍝ INIT GLOBAL ERROR SIGNAL
...
[5]     M1←READMAT 'matrix-1.txt'  ⍝ READ FROM FILE
[6]     →(0≠G_ERROR)/ER01         ⍝ I/O ERROR
[7]     M2←READMAT 'matrix-2.txt'  ⍝ READ FROM FILE
[8]     →(0≠G_ERROR)/ER01         ⍝ I/O ERROR
[9]     →((¯1↑⍴M1)≠1↑⍴M2)/ER02  ⍝ CHECK PROPER RANK OF MATRICES
[10]    RES←M1+.×M2               ⍝ PRODUCE MATRIX PRODUCT
[11]    'MATMULT.TXT' ALU⍙WRITEMAT RES
[12]    →(0≠G_ERROR)/ER04         ⍝ PROBLEMS WRITING INVERTED MATRIX
[13]    HOST ALU⍙SYSPATH,G_DIRSEP,'alu3-sigok.bat' ⍝ TELL RESULT
[14]    →0
[15]   ⍝ *** MESSAGES AND CODES ***
...
    ∇
```

Ein Beispielablauf für die Matritzenmultiplikation:

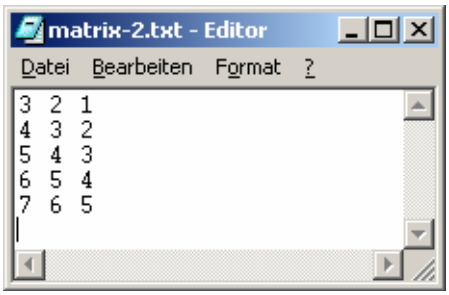
Gegeben sind in den Dateien „matrix-1.txt“ und „matrix-2.txt“ die Werte





Die Reihenfolge, in der die Dateien erstellt werden, ist beliebig. Wenn alle Daten vorhanden sind, kann die Verarbeitung veranlasst werden, in dem –für dieses Beispiel- eine leere Datei mit dem Namen doit.txt in das Verzeichnis gestellt wird. Natürlich könnte auch ein anderer Dateiname als doit.txt verabredet werden.

Das Ergebnis der Verarbeitung mit demProgramm ALUAPPL_MATMULT ist



Produktionsmatrizen

In diesem Teil geht es um das Verarbeiten von drei Arrays. Es wird als Beispiel das Verarbeiten von Produktionsmatrizen behandelt. Dabei soll bei bekannten Beziehungen zwischen Rohstoffen und Zwischenprodukten sowie der Beziehung von Zwischenprodukten zu Endprodukten bei gegebenen Endproduktmengen die Mengen der jeweiligen Rohstoffe ermittelt werden. Das Problem ist:

Problem: Produktionsmatrix auswerten

Gegeben:

- Beziehung zwischen Rohstoffen und Zwischenprodukten
  Datei *matrix-r2z.txt*
- Beziehung von Zwischen- zu Endprodukten
  Datei *matrix-z2e.txt*
- Vektor an zu erzeugenden Endprodukten
  Datei *vektor-ep.txt*

Gesucht:

- Menge der benötigten Rohstoffe
  Datei *prodmat.txt*

Es werden hier also drei Arrays (zwei Matrizen und ein Vektor) verarbeitet. Da diese in einer beliebigen Reihenfolge gespeichert werden können, wird an dieser Stelle wieder deutlich, dass es eine wesentliche Vereinfachung ist, den Start der Verarbeitung an das Vorhandensein einer Triggerdatei zu binden. In der Funktion ALU_PM wird wieder zunächst das allgemeine Umfeld für die Verarbeitung ermittelt (ALU3_INIT). Der anwendungsbezogene Teil,

- das Einlesen der Daten aus den Dateien „matrix-r2z.txt“, „matrix-z2e.txt“ und “vektor-ep.txt“,
- die Verarbeitung und
- das Bereitstellen des Ergebnisses in der Datei „prodmat.txt“

geschieht in der Funktion ALUAPPL_MATMULT.

```
∇ALU_PM[⎕]∇
[0]    ALU_PM
[1]      RC←ALU3_INIT
[2]      RC←ALUAPPL_PRODUKTIONSMATRIX
     ∇
```

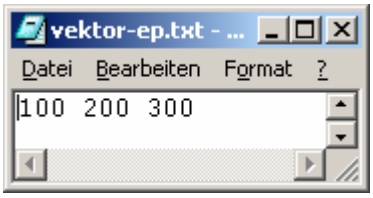
Bei der Lösung des Problems mit APL ist besonders interessant, dass die Matrizenmultiplikation unmittelbar mit einer eingebauten Funktion lösbar ist (Zeile 14).

```
          ∇ALUAPPL_PRODUKTIONSMATRIX[⎕]∇
[0]     RC←ALUAPPL_PRODUKTIONSMATRIX;M1;M2;M3;RES;R;G_ERROR
[1]     G_ERROR←RC←0              ⍝ INIT GLOBAL ERROR SIGNAL
...
[5]     M1←READMAT 'matrix-r2z.txt'   ⍝ READ FROM FILE
[6]     →(0≠G_ERROR)/ER01         ⍝ I/O ERROR
[7]     M2←READMAT 'matrix-z2e.txt'   ⍝ READ FROM FILE
[8]     →(0≠G_ERROR)/ER01         ⍝ I/O ERROR
[9]     M3←,READMAT 'vektor-ep.txt'   ⍝ READ FROM FILE
[10]    →(0≠G_ERROR)/ER01         ⍝ I/O ERROR
[11]    M3←((⍴M3),1)⍴M3
[12]    →((¯1↑⍴M1)≠1↑⍴M2)/ER02   ⍝ CHECK PROPER RANK OF MATRICES
[13]    →((¯1↑⍴M2)≠1↑⍴M3)/ER02   ⍝ CHECK PROPER RANK OF MATRICES
[14]    RES←M1+.×M2+.×M3               ⍝ PRODUCTION MATRIX
[15]    'prodmat.txt' ALUΔWRITEMAT RES
[16]    →(0≠G_ERROR)/ER04         ⍝ PROBLEMS WRITING INVERTED MATRIX
[17]    HOST ALUΔSYSPATH,G_DIRSEP,'alu3-sigok.bat' ⍝ TELL RESULT
[18]    →0
...
[27]    →0
     ∇
```

Ein Beispielablauf dazu:
Gegeben sei die Beziehung Rohstoffe zu Zwischenprodukten in der Datei „matrix-a2z.txt“:

die Beziehung von Zwischenprodukten zu Endprodukten in der Datei „matrix-z2e.txt:

sowie die Anzahl der zu fertigenden Endprodukte in der Datei „vektor-ep.txt“:



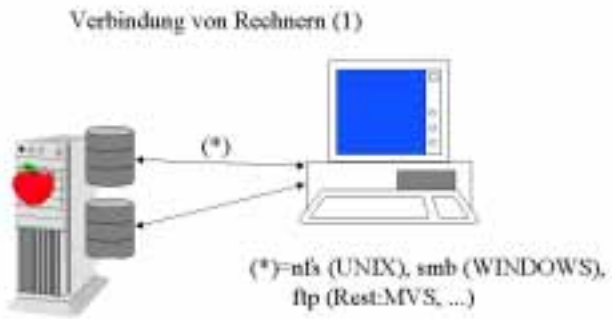
Die Reihenfolge, in der die Dateien erstellt werden, ist beliebig. Wenn alle Daten vorhanden sind, kann die Verarbeitung veranlasst werden, in dem –für dieses Beispiel- eine leere Datei mit dem Namen doit.txt in das Verzeichnis gestellt wird. Natürlich könnte auch ein anderer Dateiname als doit.txt verabredet werden.

Das Ergebnis der Verarbeitung mit obigem Programm ALUAPPL_PRODUKTIONSMATRIX ist in der Datei „prodmat.txt“:

3. Arbeitsweise im Netzwerk

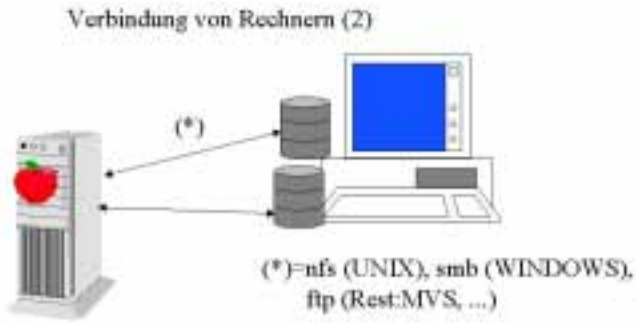
Die beschriebenen Funktionen können über Computernetzwerke rechnerübergreifend zur Verfügung gestellt werden. Dabei gibt es im wesentlichen zwei Möglichkeiten.

Zum einen können Speicherbereiche für den Zugriff von anderen Rechnern freigegeben werden. Unter Windows würde man etwa die Verzeichnisse, in denen die Arrays abgelegt werden sowie die Ergebnisverzeichnisse, welche in unseren Beispielen hier identisch sind, für einen Zugriff von außen freigeben.



Neben der vorstehend abgebildeten Vorgehensweise kommt eine zweite Variante in betracht, bei der APFS Verzeichnisse auf fremden Rechnern benutzt. In diesem Fall werden die Arrays in Verzeichnissen auf fremden Rechnern gespeichert. Dort werden die Verzeichnisse, beispielsweise bei Windows Rechnern, freigegeben, so dass auf die Verzeichnisse und Arrays von dem APFS Rechner aus zugegriffen werden kann.

Dieser Ansatz sieht bildlich wie folgt aus:



In beiden Fällen ist es grundsätzlich möglich, ein Array Processing File System zu realisieren. Der Ort, an dem die Daten tatsächlich gespeichert werden, ist eher zweitrangig.

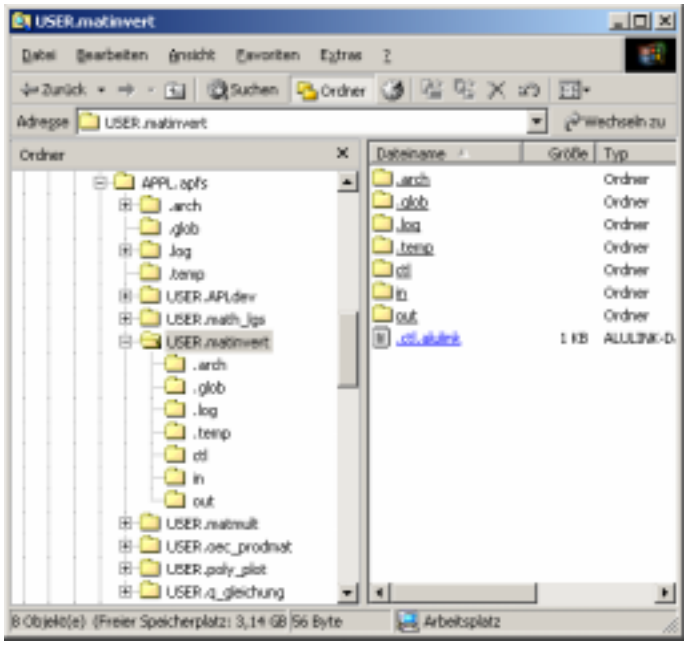
Zusätzlich kann die Verarbeitung des numerischen Teiles auf einen anderen Rechner delegiert werden. Bei mathematischen Problemen, welche sehr viel Rechenzeit verlangen, kann die Array Verarbeitung auf mehrere Rechner verteilt werden. Dies geschieht beispielsweise dadurch, dass die (auf den Abbildungen links dargestellten) APFS Rechner vervielfacht werden. Damit ist eine sehr einfache und flexible Skalierbarkeit für die numerische Verarbeitung gegeben.

4. Darstellung in einem Verzeichnisbaum

Es ist offensichtlich, dass es speziell bei der rechnerübergreifenden Verarbeitung von Arrays ein großes Problem darstellt, die relevanten Verzeichnisse aufzufinden und nach den darin enthaltenen Eingabewerten zu suchen.

In diesem Abschnitt wird gezeigt, wie sich das Array Processing File System auf einfache Weise in einem normierten Verzeichnisbaum darstellen lässt. Der normierte Verzeichnisbaum hat den Vorteil, dass er mit einfachen Methoden sehr effizient durchsucht werden kann. Die Verknüpfung zu den tatsächlich vorhandenen Verzeichnissen geschieht mit Verzeichnisverweisen.

Die oben angeführten Beispiele zur Arrayverarbeitung können in einem normierten Verzeichnisbaum wie folgt darstellt werden:



Für jedes der genannten mathematischen Probleme ist ein Verzeichnis vorhanden. Da diese jeweils gleichartig weiter unterteilt sind, ist es einfach und mit sehr effizienten Methoden möglich, den Verzeichnisbaum nach dem Vorhandensein von Dateien (hier die leere Triggerdatei) zu durchsuchen und dann die entsprechende Anwendung mit allen erforderlichen Parametern zu starten.

Wenn es gewünscht ist, dass die Eingabewerte des Array Processing File Systems an einer anderen Stelle gespeichert werden sollen, so kann dies mit einem Verzeichnisverweis, hier „.ctl.alulink“ realisiert werden. Damit kann die Schnittstelle zu anderen Dateisystemen auf dem eigenen Rechner oder auf fremden Rechnern flexibel implementiert werden.

Ein weiterer Vorteil der normierten Verzeichnisstruktur ist, dass für jede der mathematischen Funktionen in dem dazugehörenden „.glob“ Verzeichnis alle erforderlichen Konfigurations Informationen hinterlegt werden können, um die entsprechende Anwendung auszuführen.

5. Ausblick

Mit dem Array Processing File System wurde ein Ansatz gezeigt, wie mathematische Funktionen zur Array Verarbeitung in dem Dateisystem dargestellt werden können. Zusammen mit einem normierten Verzeichnisbaum ist eine effiziente und flexible Verarbeitung möglich. Darüber hinaus ist es möglich, auf Verzeichnisse von fremden Rechnern zuzugreifen oder die Verzeichnisse des Array Processing File Systems für andere Rechner in einem Netzwerk zur Verfügung zu stellen.

Mit der Nutzung von Verzeichnissen in einem Netzwerk kann die numerische Verarbeitung auf andere Rechner delegiert werden. Falls erforderlich, können die Rechner, welche die Verarbeitung für das Array Processing File System betreiben, vervielfacht werden, so dass eine einfache Skalierbarkeit der Methode gegeben ist.

Wenn das Array Processing File System in einem normierten Verzeichnisbaum dargestellt wird, ist es leicht möglich, das Arbeitsumfeld für die Anwendung zu standardisieren. Neben Methoden zum Bereitstellen der Arrays für die Verarbeitung in dem numerischen Teil sowie die Ausgabe der Ergebnisse sind zusätzlich weitere Funktionen wie das Ermitteln der relevanten Verzeichnisse standardisierbar. Diese Informationen und Verarbeitungsschritte sind als Hilfsfunktionen unabhängig von der konkreten Anwendung realisiert. Wenn diese Hilfsfunktionen die Besonderheiten unterschiedlicher Betriebssysteme beachten, so können die Funktionen des Array Processing File Systems plattform unabhängig realisiert werden. Sie sind dann ohne Änderung auf verschiedenen Rechnerplattformen, wie Unix, Linux und Windows lauffähig.

Kontakt:

Autor:
Dr.Reiner Nussbaum
mail: reiner@nussbaum-gift.de

Dyalog Version 11.0

Morten Kromberg

With the addition of *classes* and the ⎕NEW system function for creating *instances* of classes, version 11.0 is the first version of Dyalog APL with core language support for writing *object oriented* applications. However, objects are not entirely new to Dyalog APL – the first objects appeared in the language around 1990.

The first recognisable „objects" were introduced into Dyalog APL when the Windows GUI interface was added in version 6.3 – in the early 1990's. Apart from the word *property* to describe a „variable which was a child of a GUI object", object terminology never really caught on in the APL community – but whether we knew it or not, GUI elements like forms, labels or buttons were in fact *classes* that the APL developer could create *instances* of, and these instances had *properties* and *methods*.

The new objects were created and manipulated using a set of new system functions,⎕WC for the creation of objects, ⎕WG and ⎕WS for getting and setting the value of properties, and ⎕NQ for triggering events and calling methods. The original syntax required the names of objects and their members to be given as strings. For example, the statement

```
'MyForm' ⎕WS 'Caption' 'Hello World'
```

… would set the caption of MyForm. Over the next few years, Dyalog APL learned to recognize COM and Microsoft.Net classes and objects. Within APL, we were able to create *namespaces*, which were objects in that they contained collections of variables and functions. With namespaces came the familiar (to some) dot notation which APL shares with most other object-oriented languages, allowing the above expression to be shortened to:

```
MyForm.Caption←'Hello World'
```

… which is an obvious improvement in terms of readability. Finally, it was realized that references to objects could be stored in arrays, and the dot notation could be extended to allow nested name references, allowing expressions like:

```
Btns←MyForm.(Button1 Button2 Button3)
Btns.(Caption Posn)←
    ↓('Save' 'Cancel' 'Help'),[1.5] 100,¨ ¯40+50×⍳3
```

Namespaces were different from other objects in that they were not based on a *class* definition: Each namespace was created empty and could be filled with anything the APL developer desired. One could perhaps argue that this is „how APL objects should be", but there are situations where a blueprint makes ideas clearer and solutions simpler. In particular, we often want to share the same code between several namespaces without actually having several copies of the source code.

In addition to using objects within APL, the need for describing APL components in terms which are easily understood by users of other languages has become very important as standards like COM and .Net have made it practical to craft solutions from components written in more than one language. Although it has been possible to do this in Dyalog APL for several releases, it was clear that this could become more straight-forward if it were possible to define classes in APL.

The main feature of version 11.0 of Dyalog APL is the introduction of *classes* as part of the APL language. Classes can be used to create *instances*. In a nutshell, a class definition defines the functions and variables which will be available from within each instance of the class. For example, the following class definition defines a class for which every instance will contain the variables Name and Price (and sets default values for them).

```
:Class Product
    :Field Public Name←''
    :Field Public Price←0
:EndClass
```

This allows:

```
      p1←⎕NEW Product ⋄ p2←⎕NEW Product

      (p1 p2).(Name Price)←('Widget' 100)('Gadget'
150)
      products←p1 p2
      disp ⊃products.(Name Price)

      ┌→──────┬───┐
      ↓Widget│100│
      ├──────→┴~──┤
      │Gadget│150│
      └──────→┴~──┘
```

As this example shows, the simplest use of classes is to name the „leaf elements“ of an array, which can be used to increase readability and maintainability. Care has been taken to ensure that arrays of instances are „proper“ APL arrays. For example, for classes in which the default value of a field is defined, empty arrays of instances of that class can obey the same rules as nested arrays with respect to empties:

```
5↑(0/products).Price
0 0 0 0 0
```

(if you extract the Price property from an empty array of products, you get an empty *numeric* vector).

One of the important aspects of classes is the way they promote reuse of existing components. An existing class can easily be extended without modifying the original source code. A good example of this is the way a class can derive from one of the built-in GUI classes, which are not even written in APL. The following class provides a custom Form object, with user-defined defaults for selected properties:

```
:Class MyForm : 'Form'

    ∇ Make args
     :Access Public
     'Logo' ⎕WC 'BitMap' 'dyalogsmall.bmp'
     :Implements Constructor :Base args,
        ('Coord' 'Pixel')('Picture' ('Logo' 0))('BCol' (255 255 255))
    ∇

:EndClass
```

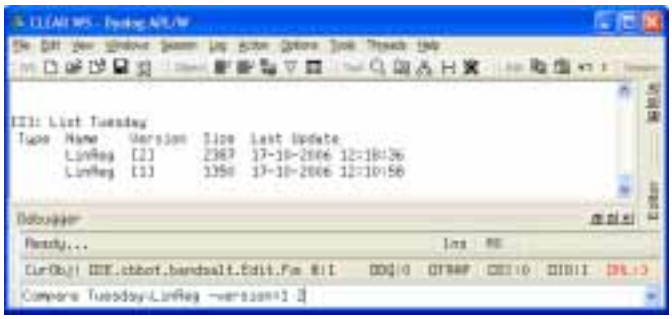
This class has no fields or properties, it only contains a constructor function which is called when an instance is created, and adds a little information before passing the arguments to the constructor of the base class. The : ‚Form‘ at the end of the first line of the above class declares that MyForm is built upon the built-in Form class. Using a class like the above as a replacement for the built-in Form class allows easy changes to application-wide defaults. Apart from having different defaults, it behaves exactly like the built-in class from which it is *derived*:

```
f1!⎕NEW MyForm (('Caption' 'Hello Germany')('Size' (100 300)))
f1.(b1!⎕NEW∇Button∇(('Caption∆ 'Press Me')('Posn'(70 240))))
```

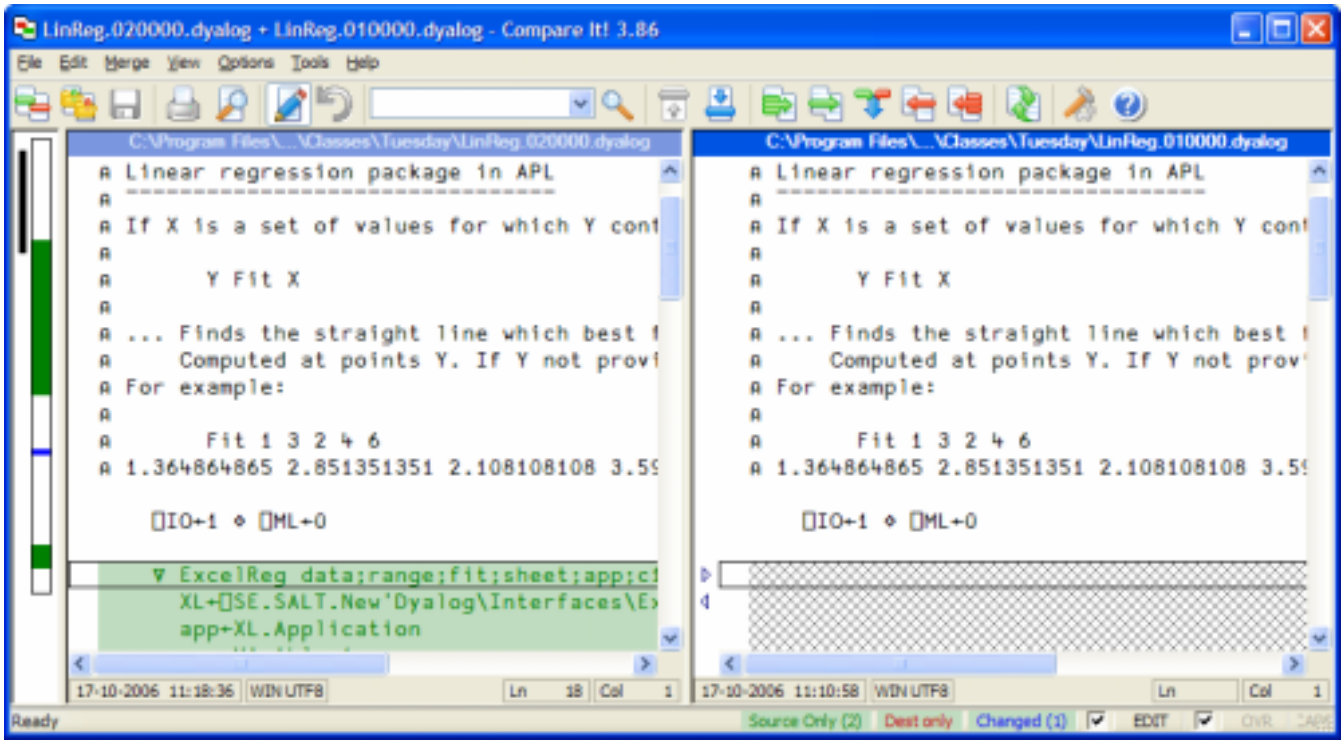
The development of Version 11.0 has taken more than two years, and is the latest step in a process which has been under way for more than a decade. Adding objects in a way which allows application of the underlying array language to objects has been quite a challenge. There is not nearly enough space in this article to to give a proper introduction to the use of classes in Dyalog APL – if you would like to learn more, see the tutorials which are included with version 11.0, available on the Dyalog and Vector web pages, or the OO Tutorial series which will appear in future issues of this publication.

Source Code Management

One of the effects of working with classes in APL which does deserve separate mention is that classes quickly lead development in the direction of storing classes (or namespaces) in Unicode text files rather than in a workspace. Classes can be loaded and unloaded on demand. The screen shot below shows an experimental command line extension to the Dyalog session, which provides development tools based on script files:



It turns out that rapid prototyping is perfectly feasible using scripts, as the scripts can be updated automatically when a function is edited, if you desire. The really big advantage of scripts is that third part tools can be brought to bear on APL systems. The Compare command which has been typed at the bottom of the previous screen shot produced the following display:



The above screen is being provided by a product called *Compare It!*, a $29 download from the Internet. It knows nothing about APL but is able to compare and merge Unicode files. Other tools which are immediately applicable to APL applications in script form are popular source code management systems like *SubVersion*. It is thus possible to switch easily between a dynamic prototyping approach and a controlled environment.

Educational and Non-commercial Licenses

The object-capable version 11.0 provides an opportunity to present Dyalog APL to new users in ways which will immediately appear more familiar to them. In order to make APL really attractive to new users, it also needs a comprehensive library of tutorials and code samples – and new educational materials. We will be working on these in 2007, but we know that it will take a little time.

We have decided to make Dyalog APL for Windows available immediately at no cost to educational users – and at low cost (£50) for non-commercial use. A similar Linux license will be available in early 2007. Educational users are required to submit an annual report on the use of Dyalog APL. Non-commercial use includes experimental use within commercial organizations: The only restriction on the non-commercial license is that it not be used to actually run applications or provide services.

Educational and non-commercial licences can be ordered on line at http://www.dyalog.com/download-zone.htm.

Other Pricing Changes

We have simplified our pricing, with three goals in mind:

- *To make it easier and cheaper to get started using APL:* The educational and non-commercial licenses are part of this strategy, and it is now also significantly cheaper to buy your first commercial license.
- *To ensure that successful APL-bases businesses contribute to investment in APL products and services:* All new licenses have support and upgrades built in. If APL is used to build applications for resale, or multi-user applications for internal use in an organization, a run-time license is required (several pricing models for runtimes are available).
- *To bring Windows and Unix pricing into line:* The run-time fee for Windows is part of this strategy; Unix users have always had to pay for run-time licenses.

The new prices take immediate effect for new business. Existing licenses may continue to pay for optional support under the old terms. The run-time license has caused concern to some: If, after looking at the pricing options, you feel that the run-time license is incompatible with your business model, please contact your Dyalog representative to negotiate an alternative. It is NOT Dyalogs intention to force any of our customers to change their business model in order to adapt to our pricing. See the Dyalog web page for more information on the new pricing.

Future Direction

The next release will be v11.1, and is scheduled for release in the 2nd half of 2007. The defining feature of this release will be the extension of the character data type to allow the use of Unicode characters in the same way as existing character variables. This will not only allow the use of nearly all of the worlds human languages in APL applications, but should also go a long way towards solving the remainig practical problems with the use of APL symbols.

Version 11.0 has almost 200 pages of Release Notes. Apart from Unicode, v11.1 is likely to have little in the way of visible language enhancements. We have almost doubled the size of the language development group in the last 12 months, and we will be spending time bringing the new recruits up to speed on the interpreter. The addition of one or two new datatypes is the perfect opportunity to go through the interpreter together with the new team and – amongst other things – apply more modern algorithms to various parts of the system (we expect to focus on searching and indexing in particular).

We also have a couple of new products in the pipeline, which will be sold separately from the language interpreter: A file server which will deliver component- and native file services over TCP/IP, and provide higher throughput and security than is possible when sharing files over a LAN or WAN – plus tools for writing high performance APL Servers both within and outside of the Microsoft.Net framework.

In addition to product development, we intend to make a significant investment in the production of new educational materials, tutorials and code samples, in 2007 and beyond. We intend to establish a repository of open-source tools, hosted on the Dyalog web site.

For more information about our direction, please visit the Dyalog web page, in particular the *Statement of Direction* under *About Us*, and the collection of presentations and course materials from the 2006 User Group Meeting, where you will find a presentation titled *Road Map 2006*.

Kontakt:

Autor;
Morten Kromberg, mkrom@dyalog.com

The Evolution of Array Programming

Dr. James A. Brown

Even the earliest computer programming languages, for example FORTRAN, had the concept of an array as a way to hold data and automate what people did by hand with tables of numbers. Subsequent languages like PL/1, Algol, Basic, Ada. even current languages like Java, C++, Visual Basic and C# have arrays and other collection classes. These languages use arrays to hold data items and then you use programming constructions to iterate over the data: DO loops, for loops, while loops, foreach loops and other iterators.

APL (A Programming Language), invented by Falkoff and Iverson in the 1960's was in some sense revolutionary in that it used the array as the fundamental unit of computing by defining operations that took arrays as arguments and produced arrays as results.

This paper traces the evolution of APLs array programming paradigm and illustrates how it has evolved and influenced modern computing.

SmartArrays® is shown to be a modern offering which captures and enhances the capabilities of APL. An example of array computing using SmartArrays in the insurance industry is discussed

Introduction

When computers were first invented, only the few could program them. Programs were entered with switches or plug boards to specify the instructions to be executed. John Von Neumann[1] revolutionized the use of computers with the invention of the stored program. Now the program was easily changed and acted like virtual switches. Assembly languages were invented to allow programming using alphanumeric representations of the machine language instructions which, with a program called an Assembler, could be translated into the bits representing the machine language of the target machine.

What was needed next was a higher level notation that could express algorithms in a more human friendly manner. John Baccus[2] and a team of others created a way to write formulas in a higher level notation and then translate what they had written into a computer program that would evaluate the formulas. Thus, one of the earliest notations given the title „programming language" was Baccus' FORTRAN (FORmula TRANslating System.

Other languages providing ever more powerful constructs were invented like PL/1, Algol, and Basic. These languages extended the paradigm of FORTRAN to become ever better ways to program computers.

APL was and is revolutionary because of its very genesis. It was not designed as a notation for programming computers. It was designed as a notation for the description of algorithms. Iverson used it in lectures on „Automatic Data Processing"[3] at Harvard from 1955 to 1960. He noticed that the algorithms for different disciplines shared many common features. He distilled these general concepts into a notation described in his seminal work „A Programming Language"[4] where the word „Progamming" did not refer to computer programming but rather to Linear programming (Al-

1 Davir Ritchie, The Computer Pioneers, 1986. Simon & Shuster, Inc., New York, Chapter 9.

2 A History of Computing in the Twentieth Century: A Collection of Papers by Los Ala International Research Conference on the History of Computing, 1980)

3 Frederick Brooks, Kenneth Iverson , Automatic Data Processing (ISBN: 0471106054)

4 Kenneth Iverson, A Programming Language, John Wiley, 1962

gebraic manipulations). After Falkoff, Iverson and Sussenguth used the notation to describe the architecture of the IBM System/360[5], the notation came to the attention of Larry Breed, then a student at Stanford University, who over time together with Richard Lathwell, Roger Moore, and many more people whom I must unfairly not mention, produced an interactive implementation of a subset of „Iverson Notation". This implementation was complete with the use of the mathematical symbols Ken used in his original writings, on the IBM 360. It used the IBM Selectric typewriter with a special type ball for input and output. When this writer was at Syracuse University in the late 1960's, APL type ball were often stolen to be worn as love beads by the coeds. (There was an earlier implementation by Breed and company written in FORTRAN but it is not possible to give a complete history in this paper.) I'm told that Adin Falkoff was the first to call this incantation of Iverson Notation „APL".

While modern languages do not have arrays as their intrinsic types, they do have the possibility of using constructed libraries (subroutines, class libraries, and such) to provide the functionality of array programming. Some languages really treat the supplied class libraries as part of the language. For example,Microsoft's C# language automates many common programming requirements with arrays and DataTables.

Most languages are designed to be efficient ways to program computers and early languages were processed into assembly language programs which in turn were processed into the machine code which could be run on the computer.

Today languages define virtual machines (Java) or byte codes (C#, Visual Basic, and so on) so that one implementation of the language, and therefore the programs written in those languages, can run on multiple computing platforms by porting the virtual machine or byte code executor.

At one time, this writer was a member of the Computer Sciences Accreditation Board (CSAB)[6] which is an organization for accreditation of programs in computer science, information systems, and software engineering. At each of the campuses I visited, I checked what texts were used to survey programming languages. A common book I found at several universities contained this single paragraph on APL (in paraphrase):

> *„APL is perhaps the most powerful array program languages ever invented. It will not be discussed further here because it had so little influence on other offerings."*[7]

If fact, APL has had a profound influence on the evolution of programming as evidenced in „The Origins of Today's OLAP Products"[8] including EXCEL and EssBase. This document credits APL as the beginning of Online Analytical Processing

What makes APL Important?

Perhaps the most important feature of APL was already stated: it was not designed for computers, it was designed for the description of algorithms. It means the user of APL deals with the problem that he wishes to solve not the mechanism of it's solution. If you ask COBOL users what they do, they would probably say „We're COBOL programmers". If you ask APL users what they do, they would probably answer „I am an engineer" or „I am an actuary". APL is what they use not what they do.

More specifically, important features of APL are:

1. Arrays are the unit of computation.

   You can often write a formula in a single line that except for syntax, matches the formula you are trying to implement. This is not surprising since the formulas came first and APL was modeled on the needs of the formulas

2. Data is dynamic

   You do not declare the length of collections. If you have a list of 50 numbers, you have a 50 item vector. The data determines the length not the programmer.

3. Data is weakly typed

   APL only has two data types: number and characters. Logical values are represented by the numbers 0 and 1. If your data is small and has no fractions, it can be stored in the computer as integers, If your data is large or has fractions, it can be stored in the computer as real numbers. If your data has imaginary components, it can be stored in the computer as complex numbers. The programmer does not make these decisions. The data determines how it should be stored. The programmer does not have to worry – „Will my integer fit in the machines integer format?"

5 Falkoff, Iverson, Sussenguth, A formal Description of System/360, IBM Systems Journal Vol3, #2&3, 1964
6 CSAB Computing Sciences Accreditation Board - http://csab.org/
7 I have not been able to locate this book for inclusion in the references.
8 The OLAP Report – the origins of today's OLAP products - March 19,2001, http://www.olapreport.com/origins.htm Copyright 2001, Business Ingelligence Ltd.

(A question that has different answers on different hardware platforms.)

4. Data is nested.

   An array is a rectangular collection of numbers, characters and other arrays.

5. Operators can apply array functions in organized ways.

   APL distinguishes between functions (that operate on arrays) and operators (that operate on functions. For example, the mathematical operation of Summation is often represented by this Mathematical expression:

$$\sum_{i=0}^{n-1} A_i$$

   But if you think of summation as the conceptual operation „add up this list of numbers“, APLs notation for it is much cleaner

```
+/A
```

   The pseudo variables „i and „n“ only clutter up the concept. If „A“ is an „n“ item vector, the addition function is applied „n-1“ times. The structure of the data determfsines how many time operations are applied – not constructions that the programmer (or mathematician) write. This makes programs shorter, easier to understand, and easier to modify.

6. APL is interpreted

   APL expressions are parsed and evaluated at execution time. While there are APL compilers, they typically put restrictions on the kinds of expressions that can be used. This interpretation used to cause people to think that APL must be slow because it is not compiled. That was never true of APL programs written to use its array power because each operation of an expression could operate on so much data. Because APL programs are small, the interpretive overhead is negligible. Now that Java is a popular language and Java programs run on a virtual machine, this kind of overhead is no longer a concern to people.

   It's interesting that this paper is written in anticipation of the 40th anniversary of the first APL workspace (1 CLEANSPACE) and a program that someone wrote 40 years ago will run on any modern APL system today without a recompilation. Can any other programming language make that claim?

7. Expressions can be evaluated interactively

   APL expressions are immediately executed so it is possible to play with your data and discover the algorithm. If you already know exactly what you need to do, this is not important but that is not a common occurrence.

8. APL is symbolic

   APL uses mathematical looking symbols to represent its operations. This makes APL expressions very short and enables the recognition of common idioms at a glance.

   Ultimately, the symbolic nature of APL may be one of the problems with the notation that limit its growth. APL programs look more difficult than they are when compared to, for example, a program in Visual BASIC. But this is because one line of Visual BASIC says so little. Furthermore, APL requires special fonts (no longer a problem) and special keyboards (still a problem).

All this is important because it empowers the person who programs in APL. A single person can implement a significant algorithm and deploy it to others in the company. The most successful companies that use APL make sure this is controlled and accomplished with good software engineering practice including documentation. Companies that allow undisciplined deployment of APL applications find years later that they have applications that are not maintainable and cannot be enhanced easily. APL gets a bad reputation at these companies.

What comes next?

When programmers have the productivity of a notation like APL, they do not respond positively to attempts to move to some newer but less productive offering. Many companies whose IT leaders did not understand the productivity of array programming mandated moving off of APL and have failed or have spent a lot of money and people to succeed.

APL is hard to beat as a programming language for algorithms but algorithms are not the only task of a business programmer. You also have to build user interfaces and you have to access databases and files and networks. You have to have version management and configuration management. The modern tools for these tasks are wonderful and productive and largely unavailable to the APL programmer.

Wouldn't it be nice if you could have it all? Array programming for algorithms; productive tools for user interfac-

es; standard programming environments; standard version and configuration management.

What's needed is a platform that preserves and even enhances the array paradigm of APL while embracing the many advantages of modern application development offerings.

This writer (whose PhD Thesis was the basis for the definition of the language APL2[9]) and James Wheeler (who was director of development for the best selling workstation offering APL*Plus) began a collaboration in 1999 that led to the definition and implementation of an offering that preserves the array programming style of APL and makes it available in today's program development environments.

SmartArrays

SmartArrays is a class library that enhances the standard programming languages to give them the array capability of APL.

What are the important features of SmartArrays?

1. Arrays are the unit of computation.

   As in APL, you can often write a formula in a single line that except for syntax, matches the formula you are trying to implement.

2. Data is dynamic

   Even though SmartArrays is embedded in languages in which data is declared, a Smart Array is determined entirely by the organization and content of its data.

3. Data is nested

   SmartArrays has three data types: number, character, and string.

4. Data is weakly typed

   A Smart Array is a rectangular collection of numbers, characters, strings and other Smart Arrays. All the advantages of weak typing mentioned before are preserved.

5. Operators can apply array functions in organized ways.
   While SmartArrays does not have operators in the sense that APL does, it does have methods to which you can pass other methods as well as arrays. For example, the composite function reduce()is the analog of reduction in APL. A summation might be phrased:

   ```
   A.reduce(Sm.plus)
   ```

6. SmartArrays is compiled

   SmartArrays is a class library that is used like any other class library you might have with your host language. The class library is, however, only a thin wrapper on highly optimized methods supplied in the SmartArrays engine DLL (or .so). Thus, you have the programmer productivity of an APL-like array notation and the performance of a compiled language.

7. Expressions can be evaluated interactively

   SmartArrays supplies the SmartArrays Interactive Compiler (SAIC) which allows you to enter expressions, evaluate them dynamically, and view results.

8. SmartArrays is not symbolic

   SmartArrays is not a language. It enhances other languages and shares their programming environments and syntax

   There is another important aspect of SmartArrays that does not parallel APL

9. Host language integration

   When you choose to use APL, you are choosing to not use other languages (at least not seamlessly). With SmartArrays, you can freely intermix elements of array programming with the object oriented schemas of the host language like Java, C#, and C++. You can use variables from the host language and Smart Arrays in a single expression.

Thus, programming in SmartArrays is similar to programming in APL – you think in terms of array collections; you write array expressions; you use operators as control structures; you use idioms. But programming in SmartArrays is also similar to programming in the standard programming languages – you use the host language syntax (C++, Java, or C#); You use the traditional control structures (for, while, iterators); You define classes and instance variables and methods; you use overloading.

The marriage of these two programming paradigms provides a very powerful platform for implementation of applications. You can make use of array processing skills with-

9 James Brown, A generalization of APL, Syracuse University, 1971, RADC-TR-73-182, Technical Report, June 1973

out giving up any of the powerful facilities of the standard languages. You can, for example, use the wonderful dialog builder of Microsoft's C# and use array algorithms APL style.

Without going into a detailed specification of SmartArrays, the following is a discussion of a common computation used by insurance companies in the determining the rates that should be charged.

Loss Triangles

An insurance company will often have different lines of business – auto, homeowners, life, and so on. For each line, the company groups people by similar risk and charge higher premium to those in a group of higher risk. For this hypothetical example, we want to look at auto losses. Three risk groups are identified, for drivers of automobiles, based on primary programming method used. These groups ordered from highest to lowest risk are:

- COBOL users
- C++ Users
- SmartArrays Users

(While this assignment is not a comment on the relative merits of the various programming paradigms. it is certainly true that more COBOL programmers have died than SmartArrays programmers. Please draw your own conclusions.) In addition, customers may choose different coverages. For this example, only two coverages are offered: full and partial. Finally, the company sells insurance in two locations in order of importance - 'TX' (Texas) and 'AE' (anyplace else) and rates vary by location.

The Loss Data

For this example, the insurance company has summarized losses by the location of the accident (State), the quarter the accident occurred, the quarter a payment was made, the customer's risk group, the kind of coverage, and the amount of money paid. (While a real company would keep track of a number of quantities (reserves, legal fees, and so on), for simplicity, we will only track actual money paid by the insurance company.

For a large insurance company, the detailed loss data is probably many hundreds of millions of rows of data. The data summarized by quarters, risks, and coverages is around 100 million rows.

Here are a few rows of this summarized data:

Location	AccQtr	PayQtr	Risk	Coverage	Payment
TX	20051	20051	Cob	Full	112
TX	20051	20051	Cob	Part	867
TX	20051	20051	CPP	Full	680
TX	20051	20051	CPP	Part	958
TX	20051	20051	SmA	Full	996
TX	20051	20051	SmA	Part	876
TX	20051	20052	Cob	Full	612

An insurance company analyses this data to determine if they have charged enough premium (or too much premium) to cover the losses, their expenses, and a reasonable profit. Typically, this analysis is done separately for each risk group. Again, in a real insurance company, other categories of information would also be considered: type of vehicle (car, truck), cause of the accident, location of the accident, and so on.

What is a Loss Triangle?

Suppose you want to examine the losses caused by accidents by Cobol programmers for all coverages in Texas. You would filter the data to select only Texas and the Cobol programmers and do sums by group on the coverages. For each set of selections, in a given quarter a certain number of auto accidents occur and a certain amount of money is paid out. This gives a simple table:

		Pay Quarter			
		20051	**20052**	**20053**	**20054**
	20051	979	1661	1823	1853
Accident	**20052**	0	4151	5250	5295
Quarter	**20053**	0	0	2314	2710
	20054	0	0	0	335

You can see that the data is triangular simply because an accident that happened in 2005 second quarter (20052) had no payments in 2005 quarter 1 (20051). So the lower triangle is all zero. Also notice that as you cross a row, the sum of payments tends to level off. This is called the development of the loss. At some point, all payments for the losses in 20051 will be paid off and you would call the loss fully developed.

In a real situation, a company might track losses going back 10 or 20 years. Because each column contains the data of the previous column, you could choose to limit the amount of data you look at by just selecting every fourth column (annually).

This is not a particularly difficult computation. The example above could easily be computed by hand. It could

easily be computed in a spreadsheet or in any programming language. What makes SmartArrays a good choice for this kind of application?

What makes SmartArrays a good choice for this kind of application?

The problem with the computation is not the complexity of the algorithm; it is the size of the data. A simple program can solve the computation on a hundred or a thousand rows of data but often these programs do not scale to the tens of millions of rows of data. You can't use Microsoft EXCEL on 50 million rows of data.

It is possible to write a program in SmartArrays that will not scale but with a little care and forethought, you can write a program that will handle truly enormous amounts of data.

There are 3 steps to building an efficient program for Loss Development.

1. Preprocess the data to reduce the volume of the data by doing summarizations in advance.

2. Store the data in a form that allows subsets to be accessed quickly

3. Write the program so that only subsets of the data

Preprocess the data

A Loss Development application is not a transactional system. Data is analyzed once a year or twice a year or quarterly. It therefore makes sense to pre-process the data as much as possible annually or quarterly to make Ad Hoc analysis of the data throughout the year or quarter efficient. Loss records gathered over a few decades could measure in the hundreds of millions of records. For our example, these records can be summarized by Line of Business, accident location, accident quarter, risk, and coverage.

Payments will be also summarized by quarter. Other keys might further divide the data – cause of loss, vehicle type, and so on. Processing this volume of the detail data might be done with SyncSort on a mainframe or a similar product. The summarized data may be reduced to under 100 Million records.

Store the data efficiently

While the data is processed only a few time a year, it is probably accessed regularly as rate changes are considered in each line an location. The most efficient way to store the data for frequent access is a way that the computing system can access it quickly. This eliminates a relational data base as the repository. The computing system can most quickly access data that is directly available on a drive – local or network – if it is in a machine supported format.

Rather than just store the whole mass of data as one large table, it is efficient to partition the data into groups likely to be accessed.

In our example, the highest level key is Line of Business. Therefore, partition the data by Line of Business and put each line in a separate directory

The next highest level key is location. We can take advantage of the inherent ordering of data in an array and sort all the data by location. If the data is sorted by location, then you do not need to store the location data as one item per record. You only need to know how many rows of data there is for each location and where in the sorted data each location begins. This is trivially stored in a two-column matrix like this:

0	78
78	78

The first location starts at the beginning of the data and extends for 78 items. The next location starts at item 78 and is also 78 items long.

Again, a real insurance company that does business work wide might have hundreds of locations. Locations are normally defined by an area subject to a given set of legislation. In the US, each state has it's own regulations.

Other columns of the table for a line of business are stored as flat files of an appropriate format. As you will see, using the location matrix, it will be possible to pinpoint the data for a location trivially.

SmartArrays provides a Data Management Toolkit that can be used to prepare the data in this format. See http://smartarrays.com/downloads/whitepapers/ for some examples of using the Toolkit and other SmartArrays related discussions.

Now the prepared data looks like this:

```
-   Auto – a directory
        o   Location Matrix – small matrix
        o   Accident quarter - integers
        o   Pay quarter - integers
        o   Risk - integers
        o   Coverage - integers
        o   Pay amount – real numbers
```

```
-   Homeowner
        o   Location Matrix - small matrix
        o   Accident quarter - integers
        o   Pay quarter - integers
        o   Risk - integers
        o   Coverage - integers
        o   Pay amount - real numbers
-   Life
        o   Location Matrix - small matrix
        o   Accident quarter - integers
        o   Pay quarter - integers
        o   Risk - integers
        o   Coverage - integers
        o   Pay amount - real numbers
```

In summary, line of business is a directory, Location Matrix is a small matrix with one row per location, each of the others is a flat file containing integers except for Pay Amount which is real numbers.

This is the style of the data organization. Many further optimizations are possible that will not be used here for simplicity. For example, it would be more efficient to store quarters as quarter number (number of quarters since 1980, for example). This together with risks and coverages could be stored in a byte wide field instead of a 4-byte integer.

Access Subsets of the data

The Loss Triangle shown earlier was for Auto in Texas. To produce that triangle, we need to access all the columns for auto losses. We do not want to access data for the locations we are not processing or a line of business that is not Auto. You choose the line of business by selecting the „Auto“ directory. You use the Location Matrix to determine the first item and number of items of data for that location in the vectors.

This technique would be of minimal help if you had to read the front part of each file to get to the location of interest. SmartArrays provides a method „fileArrayRead()“ which essentially declares that a file starting at some displacement and for a given number items, and with a given data type is an array. The file is never really read. The portion of the file of interest appears as the contents of an array but it will not actually be accessed until you use it. If you have some other criteria which limit the extent of the data even further, parts of the array may never be accessed.

You only access the part you need.. This lets SmartArrays access the relevant parts of huge data in time that almost can't be measured.

Now that the data is available, we need to select the subset of the data we want – in this case Cobol programmers. You ask where the Risk array is equal to „Cob“ and you get a Boolean mask with 1 where the corresponding record is for a Cobol programmers and 0 elsewhere. In this example, all coverages are being selected so the Coverage data vector is never touched. If we wanted only records with „Partial Coverage“, we would access the Coverage file again using the location matrix to limit the extent of data and check where Coverage is equal to „Part: and get another vector of 1's and 0's. To get a mask of records that meet both criteria, 'and' together the two masks getting a 1 only for Cobol programmers with Partial coverage. If you have more attributes to further reduce the data (maybe you only care about autos and not trucks) you get more Boolean vectors and 'and' them together. In SmartArrays, these operations are really done on bits and are extremely efficient. In most languages, logical values are bytes.

Now you know which records will be used to populate the loss triangle. Use the mask to select only the rows of interest from the Accident Quarter vector, the Pay Quarter vector and the Payment amount vector.

The remaining rows may have duplicates for a given accident, pay quarter and risk because we chose to ignore coverage. SmartArrays has a „group by sum“ method called „selectUpdateBySubsbscript(S,.plus)“ which does the group by summation in a single operation. (This somewhat cumbersome name is used because it is one in a family of other potential methods.) The result of the group-by sum is the desired loss triangle.

The following section shows and discusses the code for Loss Development.

The Loss Triangle Program

The program will need the following variables:

```
string path;        //  path to line of business
string pathm;       //  various file names
SmArray LocMat;     //  location matrix
SmArray Risk;       //  Risk data
SmArray Cov;        //  Coverage data
SmArray ACQTR;      //  Accident quarter data
SmArray PayQTR;     //  Pay quarter data
SmArray Paid;       //  Pay amount;
int loc;            //  Location selected
int start;          //  First item of data needed
SmArray extent;     //  Number of data items needed
SmArray mask;       //  Mask to select risks desired
```

We want to access the data for the Auto Line of business. This data is selected by choosing the auto directory:

```
path = „c:/auto/“;
```

We want to access the data for Texas. To do this, we need to first access the location matrix. This is conceptually a two column matrix (in the following 4 means integers):

```
// Path to location matrix
pathm = pathj + „LocMatrix.sma";
 // Declare location matrix an array
LocMat = SmArray.fileArrayRead(pathm,4);

// Determine number of rows in location matrix
int rows = LocMat.getCount() / 2;

// Reshape to a matrix
LocMat = LocMat.reshapeBy(rows,2);
```

Assume locations are identified by a code and that Texas is 1 and Anyplace else is 0. The selection of location probably comes from a user interface but we set it explicitly to Texas:

```
// Select location Texas
int loc = 1;
```

The location matrix together with the location tells us what part of the data files need to be declared as arrays.
Here is code to determine the first item of data needed and the number of items:

```
// Determine start location and extent
start = LocMat.index(loc, 0).getInt();
extent = SmArray.vector(LocMat.index(loc, 1));
```

We don't need the coverage data because we want all coverages. We do need the risk data, accident quarter, pay quarter and pay amount. Here is the code to access this data only for the desired location:

```
// Access risk data
pathm = path + „cov.sma";
Risk = SmArray.fileArrayRead(pathm, 4, extent, start);

// Access accident quarter
pathm = path + „accqtr.sma";
ACQTR = SmArray.fileArrayRead(pathm, 4, extent, start);

// Access pay quarter
pathm = path + „payqtr.sma";
PayQTR = SmArray.fileArrayRead(pathm, 4, extent, start);

// Access pay amount
pathm = path + „paid.sma";
Paid = SmArray.fileArrayRead(pathm, 5, extent, start);
```

For each of the data arrays, we want only the items that correspond to Cobol programmers. Here is a mask with 1 wherever the Risk is „Cob" (assuming the data is encoded with Cobol 0, C++1, SmartArrays 2):

```
// Select Cobol programmers
mask = Risk.eq(0);
```

Select only data for Cobol programmers:

```
// Select only data for Cobol programmers
ACQTR = mask.compress(ACQTR);
PayQTR = mask.compress(PayQTR);
Paid = mask.compress(Paid);
```

For each accident quarter and pay quarter we want to sum the losses. In the following code, the numbers (4,4) could be computed values. In this example, it is known that there are 4 quarters of data. „selectUpdateBySubscript(Sm.plus" is a „Group by sum" operation.

```
SmArray forsum = SmArray.vector(ACQTR, PayQTR);
SmArray LossT = SmArray.scalar(0).reshapeBy(4,4);
LossT.selectUpdateBySubscript(Sm.plus, Paid, forsum);
```

„LossT" is the desired Loss triangle shown previously LossT.show();

```
979   1661   1823   1853
  0   4151   5250   5295
  0      0   2314   2710
  0      0      0    335
```

As was said before, this is not a numerically intensive computation (although SmartArrays loves big computations). It is only a computation on big data. Using some forethought in data organization and using the SmartArrays feature that allows you to declare part of a file as an array means that you can greatly reduce the amount of data that is ever looked at and achieve almost instantaneous results even with a hundred million rows of data.

You could write this computation in many languages using the recipe discussed here but the array paradigm of SmartArrays allowed it to be written in very few lines of code – a few lines to access the data, some selection lines using compress to select he desired data, selectUpdateBySubscript to sum by groups. A compact program that is close to the description of the problem is easier to write, easier to read, easier to maintain, easier to enhance, and easier to understand. In this, SmartArrays emulates APL.

Conclusion

APL was the first real array processing language and it began the Online Analytical Processing (OLAP) industry. APL2 extended this paradigm to more complicated collections of data. SmartArrays captures and extends the APL

array style and makes it available in the industry standard programming environments.

People who have worked with and mastered APL have a different way of looking at problems. People that deal with lots of numbers – Actuaries, Bankers, Engineers, Scientists – picture data as rectangular collections. Array processing let you express the operations you want with a minimum of programming constructions. This lets you play with your data and play with your algorithms giving precise and compact solutions to problems.

People with array skills like APL and SmartArrays may well be the designers of new generations of applications.

General References

(1) APL History http://www.thocp.net/software/languages/apl.htm

(2) Wikipedia http://en.wikipedia.org/wiki/APL_programming_language

(3) Lathwell, Mezei, A Formal Description of APL\360, Colloque APL, Institut de Recherche d'Informatique et d'Automatique, Rocquencourt, France, 1971.

(4) A personal history of APL Micheal S. Montalbano http://ed-thelen.org/comp-hist/APL-hist.html

(5) Brown, What's wrong with APL2 http://stat.cs.tu-berlin.de/APL-Berlin-2000/confprog.htm

(6) Z. Ghandour and J. Mezei, „General Arrays, Operators and Functions," IBM J. Res. Develop. 17, 335

(7) T. More, „Axioms and Theorems for a Theory of Arrays — Part I," IBM J. Res. Develop. 17, 135 (1973).

Dr. James A. Brown
CEO, SmartArrays Inc.
www.smartarrays.com

IBM® Today's APL2

Issue Number 11 News from APL Products and Services at IBM Corporation Summer 2006

APL2 Users Explore Libraries

Service Updates

Since the last *Today's APL2* newsletter, IBM has shipped two service updates for Workstation APL2 and three enhancements for mainframe APL2.

Workstation APL2

Service Level 7 was shipped in November 2005. Enhancements include:

- COPY, PCOPY and LIB external functions
- FILE, GRAPHPAK, SQL, and MATHFNS namespaces
- New Find dialog design
- Support for COM events

Service Level 8 shipped in April 2006. In addition to the Library Manager and Calls to APL2, enhancements include:

- Windows XP Visual Styles
- Session Manager use of Unicode fonts
- AP 145 arbitrary array event handlers

Mainframe APL2

- National language keyboard support is added to AP 124 by PTFs UK07724 (TSO) and UK07725 (CMS).
- UK07516 (TSO) and UK07517 (CMS) update Processor 11 to allow routine descriptor tags to be passed in the left argument of ĀNA.
- External functions COPY, PCOPY and LIB are in PTFs UK09088 and UK09089 for TSO, UK09090 and UK09091 for CMS.

The APL2 Library Manager

For some time, APL2 customers have been telling us that they would like a way to explore the APL workspaces they have saved on disk, without the need to bring each one into an active APL2 session. The APL2 Library Manager provides such a tool for Workstation APL2 users running on Windows.

The Library Manager can open saved workspaces, transfer files, and namespaces. As each file is opened, a familiar explorer style window shows lists of the file's variables, functions, and operators with their attributes or individual object definitions.

You can also use the Library Manager to compare files. Groups of object names indicate whether objects exist in only one of the two workspaces, the definitions in both workspaces are identical, or the definitions are different.

The Library Manager can be run as an independent program or called from active APL2 sessions as an external function. When run as an external function, the library manager can compare the active workspace with workspaces saved on disk. You can use the Library Manager to discover what changes you have made since you loaded a workspace.

The Library Manager also allows you to copy objects from saved workspaces and transfer files to active APL2 sessions.

You can customize the Library Manager's environment. You can control what font and colors are used, how lists are formatted and sorted, and whether IBM-supplied names are included in the displayed lists. The Library Manager can configure Windows so you can open files by clicking on them.

We hope our APL2 customers will find the Library Manager to be a very useful tool.

Calls to APL2 from APL2

With Service Level 8, APL2 has been added to the list of languages supported by the APL2 Programming Interface. Other languages supported by this interface are C, Java and Visual Basic.

The ability to call APL2 from APL2 provides some interesting possibilities for APL2 application writers. By starting additional APL2 sessions as slaves of the main APL2 session, it is possible to create an environment that is isolated from the active workspace and distribute parts of the application to those sessions while maintaining direct control over them.

The new interface is provided in two forms. Auxiliary Processor 200 is an asynchronous shared variable interface, and external function APL2PIA is a synchronous call interface. Both provide the ability to start and stop multiple APL2 sessions, manage APL objects in the sessions and execute APL expressions and functions, using a simple command syntax.

The new APL2 Library Manager is one APL2 application that is already using this interface. It starts a slave APL2 session for each workspace opened for examination, in order to isolate them from each other and its own environment.

Learn more about APL2 at: **ibm.com**/software/awdtools/apl

Die Entdeckung statistischer Methoden im Umgang mit APL

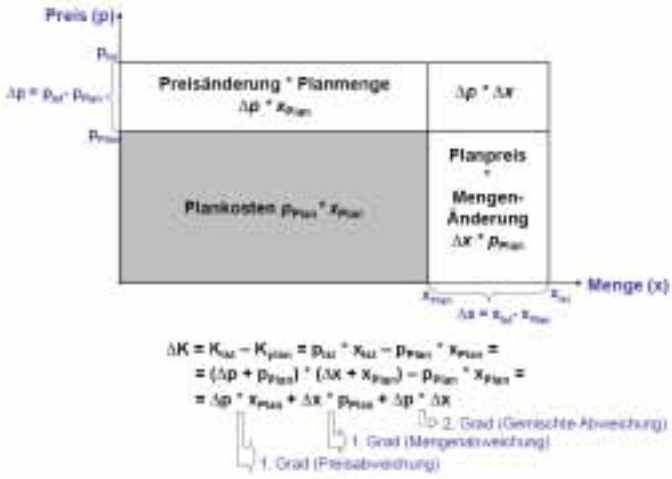
Peter-Michael Hager

Im ersten Beispiel wird gezeigt, wie Abweichungen in Kalkulationen bewertet werden können. Obwohl eine gängige Lösung bereits existierte, konnte durch APL-Tests erkannt werden, dass mit dieser etwas nicht stimmte, and und es wurden aussagekräftigere und realistischere Resultate gefunden.

Maurer-Kalkulation

Artikel	Vorkalkulation			Nachkalkulation			Differenz
	Preis	Menge	Kosten	Preis	Menge	Kosten	
Ziegel	0.75 €	165 St.	123.75 €	0.80 €	135 St.	108.00 €	-15.75 €
Zement	2.50 €	5.0 kg	12.50 €	2.70 €	6.0 kg	16.20 €	3.70 €
Sand	0.30 €	2.0 kg	0.60 €	0.40 €	30.0 kg	12.00 €	11.40 €
Total			136.85 €			136.20 €	-0.65 €

Abweichungen 1. und 2. Grades



[Quelle: Zitzmann, Betriebliches Rechnungswesen, Plankosten (Kapitel 9, Seite 31), Berner Fachhochschule]

Maurer-Kalkulation mit Abweichungen 1. und 2. Grades

Artikel	Vorkalkulation			Nachkalkulation			Differenzen durch		
	Preis	Menge	Kosten	Preis	Menge	Kosten	Preis	Menge	Mix
Ziegel	0.75 €	165 St.	123.75 €	0.80 €	135 St.	108.00 €	8.25 €	-22.50 €	-1.50 €
Zement	2.50 €	5.0 kg	12.50 €	2.70 €	6.0 kg	16.20 €	1.00 €	2.50 €	0.20 €
Sand	0.30 €	2.0 kg	0.60 €	0.40 €	30.0 kg	12.00 €	0.20 €	8.40 €	2.80 €
Total			136.85 €			136.20 €	9.45 €	-11.60 €	1.50 €

Wie zuvor, jedoch mit vertauschten Vor- und Nachkalkulations-Werten

Artikel	Nachkalkulation			Vorkalkulation			Differenzen durch		
	Preis	Menge	Kosten	Preis	Menge	Kosten	Preis	Menge	Mix
Ziegel	0.80 €	135 St.	108.00 €	0.75 €	165 St.	123.75 €	-6.75 €	24.00 €	-1.50 €
Zement	2.70 €	6.0 kg	16.20 €	2.50 €	5.0 kg	12.50 €	-1.20 €	-2.70 €	0.20 €
Sand	0.40 €	30.0 kg	12.00 €	0.30 €	2.0 kg	0.60 €	-3.00 €	-11.20 €	2.80 €
Total			136.20 €			136.85 €	-10.95 €	10.10 €	1.50 €

Die Situation verschärft sich drastisch, wenn die Kosten das Resultat dreier Faktoren sind:

Kosten = Preis × Menge × Rabattfaktor

In diesen Fällen erhält man:

- Eine Abweichung aufgrund von Preis-Differenzen,
- Eine Abweichung aufgrund von Mengen-Differenzen,
- Eine Abweichung aufgrund von Rabatt-Differenzen,
- Vier Komponenten von Mix-Abweichungen.

Dadurch, dass es nicht ungewöhnlich ist, dass die Summe dieser Mix-Komponenten größer ist, als jede zugeordnete Komponente, stellt sich die Anwendbarkeit dieser Methode in Frage. Deshalb mussten wir nach einer geeigneteren Strategie suchen.

Dieses ist die Hauptbedingung:

$a \times b + x + y = d \times e$

Einige zusätzliche Bedingungen, wie:

$a \times b + y = a \times e$
$a \times b + x = d \times b$

Was wir sonst noch benötigen, sind diese zwei Faktoren:

$p = (b \times d)^{E}$
$q = (a \times e)^{E}$

... um diese Lösung zu präsentieren:

$x = (d - a) \times (b + (e - b) \times p / (p + q))$
$y = (e - b) \times (a + (d - a) \times q / (p + q))$

Mit dem Exponenten **E=0**, wird die Mix-Abweichung halbiert:

$x = (d - a) \times (b + (e - b)/2)$
$y = (e - b) \times (a + (d - a)/2)$

Mit **E=1**, erhält der größere Verursacher den größeren Anteil:

$x = (d - a) \times (b + (e - b) \times (b \times d) / ((b \times d) + (a \times e)))$
$y = (e - b) \times (a + (d - a) \times (a \times e) / ((b \times d) + (a \times e)))$

Hier ist das selbe als dynamische Funktion:

```
ProDiff←{E←1
b←⌽a←⍺+1E¯100
d←⍵+1E¯100
h←⌽g←d-a
q←⌽p←(d×b)*E
g×b+(h×p)÷p+q}
```

Maurer-Kalkulation aus 2 Faktoren mit E=0 (Mix-Komponenten halbiert)

Artikel	Vorkalkulation			Nachkalkulation			Differenzen durch	
	Preis	Menge	Kosten	Preis	Menge	Kosten	Preis	Menge
Ziegel	0.75 €	165 St.	123.75 €	0.80 €	135 St.	108.00 €	7.50 €	-23.25 €
Zement	2.50 €	5.0 kg	12.50 €	2.70 €	6.0 kg	16.20 €	1.10 €	2.60 €
Sand	0.30 €	2.0 kg	0.60 €	0.40 €	30.0 kg	12.00 €	**1.60 €**	9.80 €
Total			136.85 €			136.20 €	10.20 €	-10.85 €

Maurer-Kalkulation aus 2 Faktoren mit E=1 (Mix-Komponenten anteilig verteilt)

Artikel	Vorkalkulation			Nachkalkulation			Differenzen durch	
	Preis	Menge	Kosten	Preis	Menge	Kosten	Preis	Menge
Ziegel	0.75 €	165 St.	123.75 €	0.80 €	135 St.	108.00 €	7.40 €	-23.15 €
Zement	2.50 €	5.0 kg	12.50 €	2.70 €	6.0 kg	16.20 €	1.09 €	2.61 €
Sand	0.30 €	2.0 kg	0.60 €	0.40 €	30.0 kg	12.00 €	**0.43 €**	10.97 €
Total			136.85 €			136.20 €	8.92 €	-9.57 €

Maurer-Kalkulation aus 3 Faktoren mit E=0 (Mix-Komponenten halbiert)

Artikel	Vorkalkulation				Nachkalkulation				Differenzen durch		
	Preis	Menge	Rabatt	Kosten	Preis	Menge	Rabatt	Kosten	Preis	Menge	Rabatt
Ziegel	0.75	165	-12 %	108.90	0.80	135	-16 %	101.52	6.82	-21.17	6.97
Zement	2.50	5.0	-5 %	11.88	2.70	6.0	-16 %	15.23	1.04	2.46	-0.14
Sand	0.30	2.0	-20 %	0.48	0.40	30.0	0 %	12.00	**1.49**	8.87	**1.17**
Total				121.26				128.75	9.34	-9.84	-7.99

Maurer-Kalkulation aus 3 Faktoren mit E=1 (Mix-Komponenten anteilig verteilt)

Artikel	Vorkalkulation				Nachkalkulation				Differenzen durch		
	Preis	Menge	Rabatt	Kosten	Preis	Menge	Rabatt	Kosten	Preis	Menge	Rabatt
Ziegel	0.75	165	-12 %	108.90	0.80	135	-16 %	101.52	6.73	-20.98	6.88
Zement	2.50	5.0	-5 %	11.88	2.70	6.0	-16 %	15.23	1.03	2.46	-0.14
Sand	0.30	2.0	-20 %	0.48	0.40	30.0	0 %	12.00	**0.41**	10.81	**0.31**
Total				121.26				128.75	8.17	-7.72	7.04

8.17 € der Differenz sind inflationsbedingt,
-7.72 € der Differenz sind Fehler des Kalkulators,
7.04 € der Differenz sind auf nicht realisierte Rabatte zurückzuführen.

```
ProDiff←{E←1
    2=⍬⍴⌽⍴⍵:⍺{
        b←⌽a←⍺+1E¯100
        d←⍵+1E¯100
        h←⌽g←d-a
        q←⌽p←(d×b)*E
        g×b+(h×p)÷p+q}⍵
    3=⍬⍴⌽⍴⍵:⍺{
        c←1⌽b←1⌽a←⍺+1E¯100
        d←⍵+1E¯100
        i←1⌽h←1⌽g←d-a
        r←1⌽q←1⌽p←a*E
        s←d*E
        pt←1⌽rs←r×s
        pu←2⌽qs←q×s
        puq←1⌽rtp←1⌽qsr←qs×r
        g×(b×c)+((h×c×qs)÷qs+pt)+((i×b×rs)÷rs+pu)+(h×i×qsr)÷qsr+rtp+puq}⍵
    ⍵-⍺}
  8 2⍕PreCalc
0.75        165.00    0.88
2.50        5.00      0.95
0.30        2.00      0.80
  8 2⍕PostCalc
0.80        135.00    0.94
2.70        6.00      0.94
0.40        30.00     1.00
8 2⍕PreCalc ProDiff PostCalc
6.73        ¯20.98    6.88
1.03        2.46      ¯0.14
0.41        10.81     0.31
```

In dem zweiten Beispiel analysiert APL rekursiv/iterativ Zahlen-Tabellen, um strukturelle Ausreißer zu ermitteln. Durch daraus hergeleitetes Einfärben erhalten diese Zahlengräber einen überzeugenden Reiz.

IQ-Test in einer Illustrierten (Level 1)

Welche Zahl passt nicht zu den übrigen?
1001 996 1000 1005 2004 998 997 999 1004 1003

```
      IQuestion←995+10?10
      IQuestion[5]×←2
      IQuestion
1001996 1000 1005 2004 998 997 999 1004 1003
      0⍕Normalize IQuestion
1 1 1 1 2 1 1 1 1 1
     Normalize←{(⍵×⍴⍵)÷+/|⍵}
```

IQ-Test in einer Illustrierten (Level 2)

507	364	1196	689	611	1235	78	1001	1014	1079
78	56	184	106	94	190	12	154	156	166
2691	1932	6348	3657	3243	6555	414	5313	5382	5727
3393	2436	8004	4611	4089	8265	522	6699	6786	7221
2457	1764	5796	3339	5922	5985	378	4851	4914	5229
2886	2072	6808	3922	3478	7030	444	5698	5772	6142
2847	2044	6716	3869	3431	6935	438	5621	5694	6059
3900	2800	9200	5300	4700	9500	600	7700	7800	8300
3471	2492	8188	4717	4183	8455	534	6853	6942	7387
936	672	2208	1272	1128	2280	144	1848	1872	1992

```
      Sales¥(10?100)Ñ.ø(10?100)
      Sales[5;5]ø¥2
      Sales
 507  364 1196  689  611 1235   78 1001 1014 1079
  78   56  184  106   94  190   12  154  156  166
2691 1932 6348 3657 3243 6555  414 5313 5382 5727
3393 2436 8004 4611 4089 8265  522 6699 6786 7221
2457 1764 5796 3339 5922 5985  378 4851 4914 5229
2886 2072 6808 3922 3478 7030  444 5698 5772 6142
2847 2044 6716 3869 3431 6935  438 5621 5694 6059
3900 2800 9200 5300 4700 9500  600 7700 7800 8300
3471 2492 8188 4717 4183 8455  534 6853 6942 7387
 936  672 2208 1272 1128 2280  144 1848 1872 1992
      0¶Normalize Sales
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1
```

Regionale Vertriebs-Umsätze

Region \ Items	Pants	Jeans	Shirts	Skirts	Jackets	Suits	Sweater	Slippers	Sneakers	Socks
Austria	507	364	1196	689	611	1235	78	1001	1014	1079
Belgium	78	56	184	106	94	190	12	154	156	166
Denmark	2691	1932	6348	3657	3243	6555	414	5313	5382	5727
France	3393	2436	8004	4611	4089	8265	522	6699	6786	7221
Germany	2457	1764	5796	3339	5922	5985	378	4851	4914	5229
Great Britain	2886	2072	6808	3922	3478	7030	444	5698	5772	6142
Holland	2847	2044	6716	3869	3431	6935	438	5621	5694	6059
Italy	3900	2800	9200	5300	4700	9500	600	7700	7800	8300
Spain	3471	2492	8188	4717	4183	8455	534	6853	6942	7387
Sweden	936	672	2208	1272	1128	2280	144	1848	1872	1992

```
Normalize⍵⍵{ ⍝ Normalize an array of arbitrary rank - Dyalog V11 version
    AxialNormalize⍵⍵{(⍹×⍺ ⍸⍹)÷[(⍸⍸⍸⍹)~⍺]+/[⍺]|⍹}
    LeftChained⍵⍵{ ⍺⍺/(⍵⍺), ⍹}
    Preprocess⍵⍵{⍹+1E¯180×0=⍹}
    Postprocess⍵⍵{⍹×1E¯160<|⍹}
    Postprocess(⍸⍸⍸⍹)AxialNormalize LeftChainedŐ Preprocess ⍹
}

Normalize⍵⍵{ ⍝ Normalize an array of arbitrary rank - Compatible version
    AxialNormalize⍵⍵{(⍹×⍺ ⍸⍹)÷[(⍸⍸⍸⍹)~⍺]+/[⍺]|⍹}
    LeftChained⍵⍵{ ⍺⍺/(⍵⍺), ⍹}
    Until⍵⍵{⍹ ⍹⍹ x⍵⍵⍺⍺ ⍹:x | ⍵⍵ x}
    Preprocess⍵⍵{⍹+1E¯180×0=⍹}
    Postprocess⍵⍵{⍹×1E¯160<|⍹}
    Postprocess{(⍸⍸⍸⍹)AxialNormalize LeftChained ⍹}Until Preprocess ⍹
}
```

Deutsche Bevölkerungsstatistik

Alter \ Jahr	1999	2000	2001	2002	2003
0	771223	766554	735755	719250	706449
1	788905	772940	772147	739547	722845
2	814425	789766	775918	773772	741235
3	799778	814674	792198	777811	775035
4	773303	799910	816966	794141	778756
5	782784	773740	802195	819000	794971
6	814607	782802	775663	803429	819102
7	833793	814593	784716	777074	803783
8	866345	834766	817082	786980	778263
9	954074	867444	837502	819359	788064
10	943186	955399	870230	840052	820427
11	970098	945001	958622	873253	841560
12	949410	972105	948491	961356	874970
13	933225	951793	975676	951535	963201
14	901858	935755	955683	979021	953449
15	900050	905014	940218	959420	981339
16	908955	904300	911412	945119	962594
17	936004	913630	910945	916538	948796
18	937447	942036	922226	917900	921679
19	950649	947452	955826	934459	927797
20	911786	963406	963199	969933	945735
21	908151	922857	978755	976442	980454
22	910085	920340	939322	992803	987005
23	905701	921590	936609	952985	1003813
24	890669	916064	936232	948860	962464
25	912320	898814	928552	946009	956147
26	931254	918245	909454	936263	951628
27	1029212	935974	927431	916169	940907

Deutsche Bevölkerungsstatistik (Fortsetzung)

28	1144718	1032886	943227	933124	919707
29	1202052	1147340	1038983	947840	935548
30	1296324	1203587	1151707	1042534	949498
31	1362258	1296851	1207656	1154077	1043319
32	1398811	1362199	1299552	1209961	1154414
33	1439408	1398313	1364225	1300709	1209537
34	1450859	1438889	1399998	1364686	1300032
35	1477274	1450005	1440128	1400187	1363600
36	1469050	1476581	1450670	1439945	1398377
37	1424872	1468164	1476899	1450555	1438447
38	1404723	1424090	1468690	1476463	1448475
39	1368060	1403988	1424674	1468226	1474533
40	1326138	1367322	1404263	1424198	1466359
41	1261066	1325466	1367630	1403660	1422470
42	1238416	1260096	1325031	1366909	1401548
43	1210553	1237357	1259536	1324022	1364641
44	1177514	1208956	1236607	1258214	1321898
45	1161536	1176030	1207862	1234722	1256051
46	1128557	1159795	1174673	1205707	1232037
47	1133521	1126482	1157765	1172503	1202886
48	1113454	1131138	1124315	1155464	1169433
49	1123726	1110862	1128577	1121436	1152162
50	1087755	1120698	1108022	1125457	1117790
51	1003191	1084091	1117385	1104667	1121792
52	947557	999443	1080585	1113405	1100655
53	834565	943614	995557	1076523	1108775
54	742176	830302	939581	991293	1071635
55	987985	737949	826037	934937	986273
56	1006682	981853	733724	821345	929925

57	981990	999971	975354	729066	816200
58	1189781	975077	992976	968455	723816
59	1258458	1181002	968067	985523	960921
60	1235197	1247770	1171255	959970	976976
61	1153916	1223925	1236424	1161033	951271
62	1078082	1142454	1212233	1224590	1150029
63	1049741	1066784	1130839	1199711	1211778
64	1012960	1037232	1054855	1118264	1186400
65	940888	999448	1024140	1042034	1104720
66	757258	926714	985463	1010252	1028050
67	752064	744532	912376	970422	995364
68	762724	738522	731607	896637	954207
69	799984	747726	724474	717779	880013
70	770179	782641	732099	709693	703388
71	758324	752048	764629	715508	693668
72	695572	738595	732999	745642	697782
73	673646	676352	718511	712768	725229
74	655313	653544	656444	696700	691416
75	595573	633548	632558	634908	673450
76	568893	574342	610897	610122	613167
77	573642	545926	551956	587216	586192
78	574718	547767	521892	528378	561897
79	537637	545651	520874	496753	503089
80	384281	506703	515478	492383	469764
81	235001	360199	475438	483677	462122
82	210078	218196	335240	442751	450653
83	214385	193387	201446	309447	408189
84	255958	195477	177109	184069	282744
85+	1635134	1612696	1540092	1451680	1374891

Kontakt:

Peter-Michael Hager
HAGER-ELECTRONICS GmbH
Hamburger Str. 97, 44135 Dortmund, Germany
mailto: hager@dortmund.netAxel Holzmüller,
eMail: axel.holzmueller@dpc.de

Abstract

The world of Software development is getting wider and wider. Applications run in a variety of environments and use components written in a variety of languages. Recent APL2 enhancements allow developers to participate more fully in this world of component use and reuse. The new APL2 Programming Interface (Calls to APL2) is a set of APIs that allows applications to start and control APL2. Calls to APL2 provides the foundation for APL2's new Interfaces with other languages, components, and environments including Java, Visual Basic, and Microsoft's Component Object Model (COM).

APL2 and Java

APL2 has two new interfaces to Java:

- Associated Processor 14 for calling Java from APL2
  Enables applications to exploit the enormous wealth of components available as Java classes such as XML parsing and infinite precision arithmetic.
- The Java interface to Calls to APL2
  Enables Java applications to call APL2 applications. It enables APL2 developers to integrate their applications in mainstream server applications such as IBM's WebSphere Application Server.

Calling Java from APL2

Associated Processor 14

- Reference and specify static fields
- Call static methods
- Instantiate Java objects
- Reference and specify instance fields
- Call instance methods

Java methods can call back to APL2

Java Language Overview

```
public class Sample
{
  public static int StaticField = 0 ;
  public static int StaticMethod(int Argument) {
        return Argument + StaticField ;
        }
  public int InstanceField ;
  public Sample(int InitialValue) {
        this.InstanceField = InitialValue ;
        return ;
        }
  public int InstanceMethod(int Argument) {
        return Argument + this.InstanceField ;
        }
  }
```

Associated Processor 14 Syntax

```
(class  'signature')  14  ⍕MA  'surrogate member'
```

class — A class identifier. If associating a static member, then class is a character vector containing the name of the class. If associating an instance member, then class is an instance integer returned by a constructor.

signature — A character vector description of the member. If the member is a field, then the signature is a description of the field's datatype. If the member is a method, then the signature is a description of the method's arguments and result.

member — The name of the Java method or field.

surrogate — The name to be used within the APL workspace.

Java Signature Rules

Java uses the following characters to describe native data types:

Code	Type
B	byte
C	char
D	double
F	float
1	int
J	long
S	short
V	void
Z	boolean

A left square bracket indicates an array.
A class name is indicated by proceeding the name with an L and following it with a semi-colon.

Java Signature Examples

'I'

An integer field

'Ljava/lang/String;'

A String field

'(II)D'

A method with 2 integer arguments and returns a double

'(Ljava/lang/String;)[I'

A method with a String argument and returns an integer array

Sample Processor 14 Associations

```
      ⍝ Associate a name with a static field
      ('Sample' 'I') 14 ⎕NA 'IncrementAmount'
1
      ⍝ Associate a name with a static method
      ('Sample' '(I)I') 14 ⎕NA 'Increment'
1
      ⍝ Associate a name with a constructor
      ('Sample' '(I)V') 14 ⎕NA 'CONSTRUCT <init>'
1
      ⍝ Call the constructor
      Instance←CONSTRUCT 34
1
      ⍝ Associate a name with an instance field
      (Instance 'I') 14 ⎕NA 'Value'
1
      ⍝ Associate a name with the instance method
      (Instance '(I)Z') 14 ⎕NA 'IsEqual'
1
```

Calling APL2 from Java

The Java Interface to Calls to APL2 supports the following features:

- Starting and Stopping APL2 Interpreters
- Creating and Deleting Workspace Objects
- Assigning, Associating, and Expunging Names
- Executing Expressions and Functions
- Using APL Characters in Java Programs
- Querying Workspace Object Attributes
- Retrieving Workspace Object Values
- Handling APL2 Errors

The APL2 Java Classes

The APL2-Java interface includes the following classes for using APL2 from Java:

Apl2interp	Make APL2 interpreter requests
Apl2object	Manage APL2 workspace objects
Apl2exception	Indicate and detect APL2 errors
Apl2cdr	Convert between workspace and CDR formats

An APL2 interpreter called from Java has the same restrictions **as** the APL2 Runtime Library.

Sample Java Programm

```
//* Import the apl2 package of classes
import com.ibm.apl2.*;
public class Sample {
   public static void main(String[] args) {
      try {
         //* Create a slave interpreter
         Apl2interp Slave = new Apl2interp(new String[]
            {"-ws","lm"}) ;
         //* Reference QuadTS
         Apl2object Time = Slave.Execute(Apl2interp.QTS) ;
         //* Convert it to a Java integer array
         int[] TimeArray = Time.intarrayValue() ;
         //* Free the workspace object
         Time.Free();
         //* Stop the interpreter
         Slave.Stop();
      }
      catch (Apl2exception Exception) {
         System.out.println("Apl2exception caught.");
         System.out.println("APL2 Exception.  Event: " +
            Exception.Type + " " + Exception.Code) ;
      }
   return ;
   }
}
```

Java Resources

The Java Tutorial:
java.sun.com/docs/books/tutorial/index.html
APL2's interfaces to Java:
APL2 User's Guide
And a new APL2 book:
APL2 Programming: Using APL2 with WebSphere

APL2, COM, and Visual Basic

APL2 has two new interfaces on Windows:

- The COM external function
  Enables APL2 applications to access Component Object Model (COM) objects including applications such as Microsoft Agent and Excel. Applications can efficiently pass arrays directly between APL2 and COM objects.
- The Visual Basic interface to Calls to APL2
  Enables APL2 to be easily called from products that support Visual Basic macros such as Microsoft Excel and Word.

Using COM Objects from APL2

The COM external function provides **access** to Microsoft Component Object Model (COM) objects.

```
[result ←]    [control]    COM    'COMMAND'    [arguments]
```

The COM function supports several commands:

QUERY	Query information about COM classes and objects
CREATE	Create instances of COM classes
CONNECT	Connect to running COM programs
METHOD	Invoke COM object methods
PROPERTY	Specify and reference COM object properties
HANDLERS	Specify and reference COM object event handlers
WAIT	Wait for COM events
RELEASE	Release references to COM objects
CONFIG	Set configuration options

COM Events Example

```
EXCEL←COM 'CREATE' 'Excel.Application'
COM 'HANDLERS' EXCEL (1 2⍴'WorkbookBeforeClose' CLOSE)
COM 'PROPERTY' EXCEL 'Visible' 1

LOOP: →(''≡RESULT←COM 'WAIT' 5)/LOOP
(OBJECT EVENT HANDLER CURSORPOS TIME NAMED)←6↑RESULT
POSITIONAL←6↓RESULT
→HANDLER

CLOSE:
COM 'PROPERTY' EXCEL 'Visible' 0
COM 'PROPERTY' EXCEL 'DisplayAlerts' 0
COM 'HANDLERS' EXCEL ''
COM 'RELEASE' EXCEL
```

Simple COM Example

```
EXCEL←COM 'CREATE' 'Excel.Application'

WB←COM 'METHOD' EXCEL 'Workbooks.Open''D:\TEST.xls'
DATA←COM 'PROPERTY' EXCEL 'ActiveSheet.UsedRange.Value'
COM 'PROPERTY' EXCEL 'DisplayAlerts' 0
COM 'METHOD' WB 'Close'
COM 'RELEASE' WB
COM 'RELEASE' EXCEL
```

Calling APL2 from Visual Basic

The Visual Basic interface to Calls to APL2 supports the following features:

- Starting and Stopping APL2 Interpreters
- Using Workspace Objects
- Assigning, Associating, and Expunging Names
- Executing Expressions and Functions
- Using APL Characters in Visual Basic Programs
- Handling APL2 Errors
- Data Conversion Between Visual Basic and APL2

The APL2 Visual Basic Functions

- StartApl2
- StopApl2
- VariantToLocator
- LocatorToVariant
- ExecuteExpression
- ExecuteMonadic
- ExecuteDyadic
- Assign
- Associate
- Expunge
- FreeLocator
- GetET
- GetMsg

An APL2 interpreter called from Visual Basic has the same restrictions **as** the APL2 Runtime Library.

Sample Visual Basic Program

```
Token = StartApl2(Array(„-ws“, „2m“))
Left = VariantToLocator(Token, Array(123, 456))
Right = VariantToLocator(Token, Array(789, 123))
Func = VariantToLocator(Token,“+“)
Result = ExecuteDyadic(Token, Left, Func, Right)
FreeLocator Token, Left
FreeLocator Token, Right
FreeLocator Token, Func
BoolRc = LocatorToVariant(Token, Result, Var)
FreeLocator Token, Result
Worksheets(„Sheet1“).Range(„B14:B15“). Formula = Var
StopApl2 Token
```

APL2 Contact Information
Email: APL2@vnet.ibm.com
BM web site - www.ibm.com/software/awdtools/apl *Downloads for* demonstration version *Library for* on-line APL2 manuals News for latest announcements *Support for* FAQ, Technotes, and Service Updates
IBM APL2 Newsgroup
news.software.ibm.com/ibm.software.apl

Workstation APL2 Version 2 Service Level 9

Nancy Wheeler, IBM APL Products and Services

In November 2006, Service Level 9 to Workstation APL2 Version 2 was made available. In addition to the usual fixes for reported problems, this service level contains several enhancements which address requirements submitted to IBM by the German customers of APL2.

Interpreter Deviation Removal

Several changes to the APL2 interpreter remove documented deviations from the APL2 Language Reference:

Display of Nested Arrays

In columns with both numeric and character items, the character items are right-justified. In columns with only non-numeric data, character items are left-justified. Previously all character items were left-justified.

Prototype extension in Disclose, Expand, Replicate and Take

When results of these operations require the use of fill items, the added items use the prototypes for the first element of the correct dimension rather than always using the prototype of the first element of the array.

Partition results with null arguments

The results from partition when one or both arguments are null are the correct depth and shape as defined by the language.

Monadic format of nested arrays

The depth of the result of monadic format of nested arrays is at most two, instead of taking the depth of the argument.

Processor 11 Names File Search

Support has been added for searching more than one names file for an external routine's descriptor. Where a single file name was previously provided, multiple file names separated by ; (Windows) or : (Unix) may now be provided.

GRDATA for Unix Systems

The AP 207 command GRDATA allows the contents of the screen to be saved and loaded using fourteen different graphics formats, including JPEG, GIF and Bitmap.

Previously the GRDATA command was only implemented on Windows. Service Level 9 adds the command to AP 207 on AIX, Linux and Sun Solaris, with the same syntax and functionality.

In addition, the GRDATA command has been reworked to eliminate the use of temporary disk files. Along with simplifying processing, this will improve performance on all the platforms.

APL2 Library Manager

The new APL2 Library Manager, added to Windows APL2 systems in Service Level 8, provides the ability to browse and compare APL2 transfer files, workspaces and namespaces.

In Service Level 9, files with non-standard extensions are supported. The APL2 file type will be determined by examination of the contents.

Two new customization options have also been added to the Library Manager. *Compare Timestamps* controls whether timestamps are included when comparing functions and operators. *Sort Using DCS* allows for an alternate sorting algorithm based on the DCS variable from the EXAMPLES workspace.

Windows Object Editor

The Object Editor now recognizes and underlines URL's found in comments. You can double-click or press Ctrl+0 to open the URL.

The new Object Editor option *Always on Top* allows you to specify that your editor window should remain the top window on your desktop.

AP 145 Extensions

Unicode support in AP 145 has been enhanced with support for Unicode data in menus, status areas and tab controls, the addition of Unicode versions of the FILEDLG, FOLDERDLG and POPUPMENU functions, and exten-

sion of the UNIMSGBOX function to support styles.

Other extensions to AP 145 in this service level include:

- Drag and drop of files onto dialogs, listviews, MLEs, treeviews, and custom controls.
- *Hover* event support
- Editing of Listview labels
- Support for pictures on menu items
- Specification of argument and result types for LOAD-API

APL2 on Windows Vista

Preliminary testing of APL2 on Windows Vista revealed several minor incompatibilities. Fixes for these problems have been shipped in Service Level 9.

Verbesserungen für APL2 V2.02 auf Großrechner

TSO PTF	CMS PTF	Beschreibung

UK07724	UK07725	AP 124 Unterstützung Nationalsprachen
		`CTL←8 0 ⍝ Deaktivieren Nationalsprachen`
		`CTL←8 1 ⍝ Aktivieren Nationalsprachen`
		`CTL←8 2 ⍝ Abfragen Nationalsprachen`
		`DAT[2] ←→ 0 Inaktiv`
		`DAT[2] ←→ 1 Aktiv`

UK07516	UK07517	Deskriptor von Prozessor-11-Routinen im linken Argument von `quadNA`
UK09088 UK09089	UK09090 UK09091	Externe Funktionen COPY, PCOPY und LIB ermöglichen programmierbaren Zugriff auf Systembefehle

Unterstützung für APL2 auf Großrechner

- Sie können online den neuesten Stand der Gesamtliste der für APL2 V2.02 verfügbaren Programmkorrekturen (PTF) abfragen:

  http://www.ibm.com/software/awdtools/apl/support.html
  - Klicken Sie dort auf "APARs" im Abschnitt "Self help" unter "Solve a problem".
  - Außerdem finden Sie dort auch "FAQs" und "Technotes"

  - Im Abschnitt ***Library*** finden Sie APL2-Online-Handbücher (in englisch)
  - Im Abschnitt *News* finden Sie die neuesten Ankündigungen

- Unterstützung per Email:
  APL2@vnet.ibm.com

APL's 40th Anniversary at IBM

The first APL workspace was saved at IBM research facility in Yorktown, New York on November 27, 1966. You can still load this workspace today in any IBM APL2 system using the command:

```
)LOAD 1 CLEANSPACE
```

To celebrate the 40th anniversary of APL, special events have been taking place at the IBM Silicon Valley Laboratory in San Jose, California. Bulletin board displays started appearing in September. A contest was held to see who could guess the topic - at first only the frame of the APL2 display logo appeared, and new clues were added each week. We gave away several APL2 T-shirts and book bags to the winners. A lab-wide party was held on November 1, with wine and cheese, congratulatory speeches, a display of APL memorabilia, and a slide show of specially created APL2 graphic displays.

In addition, we collected testimonial statements about APL and APL2 from some of our customers. These testimonials have been published on the IBM APL2 web site: www.ibm.com/software/awdtools/apl

Click on Success Stories to see the testimonials.

```
Happy Birthday APL!
```

Nancy Wheeler,
IBM APL Products and Services

APL2 can be a true gestalt

A Respectful Testimonia

APL2 has been a necessary and sufficient programming environment in my career in science for almost thirty years. My perspectives and experiences are personal, unique and certainly orthogonal to the mainstream programming directions that evolved here at The Johns Hopkins University Applied Physics Laboratory (JHU/APL). Yep, APL at APL, a double-dose. Starting in the early '80s, there was a small group of APL aficionados at JHU/APL enamored with „Iverson's Language." We used „IBM3279" terminals and were obviously much cooler than the trudging dinosaurs who carried ten thousand IBM punch cards in cumbersome 30-inch long metal drawers and toted them to the central „computer center" for compilation. My experience started with APL(1) on the IBM 370 and Amdahl mainframes in the late '70s and moved to APL2, also on a mainframe, in the mid '80s (we operated on MVS and then VM). I used APL2 to complete the compounded numerical regression analyses and graphics for my Ph.D. dissertation and then somewhat reluctantly migrated to the PC DOS OS workstation environment shortly afterwards. It turned out that the Madrid & Manchester APL2 application was a more-than-adequate and welcomed alternative to mainframe computing, which shortly thereafter became unavailable. The current Workstation APL2 product running on Windows is simply essential to the life that I have invented and nurtured.

APL2 can be a true gestalt, a liberating computational Weltanschauung for the initiated. When practicing APL2, discovery of latent recursive algorithms frequently occurs serendipitously after data are first organized into logical, multivariate nested arrays. And recursion, as we all know, is the holy grail of algorithm development. Emergence of insightful „divide and conquer" strategies becomes inevitable once the approach is redirected top-down on suitably partitioned data. This experiential pattern suggests a unique value of APL2: it can reveal a subterranean simplicity beneath surface turbulence. Thus APL2 is discovered ex post facto to be an epistemology, a way of discovering, learning and knowing about the mechanization of problem solutions through the enlightened manipulation of data. A fluid implementation with concrete results: an ethereal anti-depressant therapy for the over-burdened and undervalued. So there you have it, the truth, the whole truth, and nothing else.

I remain grateful to the IBM Silicon Valley Lab and Manchester & Madrid Centers, and the founding fathers, Ken Iverson, Adin Falkoff and Jim Brown, for conceiving, creating and enhancing an iconoclastic mind-bender of enduring value. Your persistence and longevity have been and remain invaluable to me.

Customer: Johns Hopkins University Applied Physics Laboratory
Author: Richard Stockbridge
Country: United States of America
The Intersection of APL2 and Richard D. Stockbridge

Mehr als 20 Jahre APL bei Thomas Cook

Die Thomas Cook AG ist einer der führenden Touristikkonzerne der Welt. Die Geschäftsanteile an der Thomas Cook AG werden zu jeweils 50 Prozent von der Deutschen Lufthansa AG und der KarstadtQuelle AG gehalten.

Die Thomas Cook AG wurde 1998 als Zusammenschluss von Condor und NUR Touristic unter dem Namen C&N Touristic AG gegründet. Nach Übernahme der britischen Thomas Cook Ltd wurde die C&N Touristic AG im Sommer 2001 in Thomas Cook AG umbenannt.

1985 - 86: Prototypen Kalkulationssysteme FlugNah und ABBF

Neckermann Urlaubsreisen (NUR) entwickelte Anfang der 80er Jahre des vergangenen Jahrhunderts immer mehr Bedarf nach einem Kalkulationssystem.

Dieses sollte zum einen die Preisfindung unterstützen und zum anderen die Simulation einer Reisesaison in Form einer Ergebnisrechnung ermöglichen.

Traditionell ist der Flugpauschalreisemarkt im Nahbereich durch das Massengeschäft geprägt, während der Markt der Reisen mit Auto, Bus und Bahn (ABBF) durch individuelle Angebote und Vielfalt glänzt.

Beiden ist ein enormer Varianten-reichtum zu Eigen, der aus Sicht der IT zu erhöhten Anforderungen an Rechen- und Speicherkapazität führt.

Zu dieser Zeit, PCs waren kaum den Kinderschuhen entwachsen, war es klar, dass nur ein IBM Großrechner die Aufgabe bewältigen konnte.

Da die Verfahren nach denen die Ergebnisrechnung arbeiten sollte, zunächst noch erarbeitet werden mussten, bot sich eine dynamische Interpretersprache, wie APL, gerade zu an.

So entstanden, unter Mitwirkung der Unternehmensberatung McKinsey, die Systeme FlugNah für Flugpauschalreisen und ABBF für Reisen mit Auto, Bus und Bahn.

1986 - 91: Ausbau und Pflege FlugNah, ABBF

In den folgenden Jahren mussten immer wieder Anpassungen gemacht werden, weil die Marktbedürfnisse neue Reiseprodukte und Preiskomponenten erforderten.

In diese Zeit fiel auch die Migration von VS APL nach APL2, welches durch seine neuen Sprachmittel weitere Vorteile brachtegegenüber den Standard-Programmiersprachen, wie Cobol.

1991: Neues Kalkulationssystem für Belgien (NVB)

Seine Mächtigkeit bei der Entwicklung von Anwendungen bewies APL in diesem Jahr, als es gelang, innerhalb weniger Wochen ein neues System, speziell für den belgischen Markt, von nur einem Entwickler der Firma Dittrich und Partner Consulting (DPC) entwickeln zu lassen.

1992: Umstellung von VSAM auf DB2

Der Form nach war diese Umstellung nur eine technische Maßnahme, aber ein großer Schritt in Sachen Datensicherheit.

Im Übrigen nutzt Thomas Cook seid dieser Zeit externe Cobol-Programme als Zugriffsschicht zum DB2. Diese bieten sich aus Performance- und Sicherheitsgründen an.

1994 - 1996: Ablösung von ABBF durch TABU (tageweise Buchen) und NVB durch BELI

In diesen Jahren wurden die betagten Altsysteme unter Mitarbeit der Firmen DPC und a.k.e abgelöst und zugleich der Funktionsumfang wesentlich erweitert. Gerade in Belgien zeichnete sich zu dieser Zeit ab, dass der Reisemarkt dynamischer werden würde. Die Antwort von Neckermann waren mehrere Preisteile, die heute auch in Deutschland kaum mehr wegzudenken sind.

1999 -2002: Flugkalkulation Neu

1999 war auch für das Pauschalreisesystem die Zeit gekommen. Neue Anforderungen aus dem Fernreisebereich, durch Preiskomponenten wie Frühbucher und Incentives, aber auch die Notwendigkeit der Integration des parallel entwickelten Einkaufmanagementsystems machten ein komplettes Re-Design notwendig.

Technisch sollte das neue System den neuen Technologien gerecht werden, die zu der Zeit Einzug in die IT hielten. Als Programmiersprache wurde zunächst Java gewählt, jedoch scheiterten alle Versuche, auch die APL2 Anteile durch Java zu ersetzen, an Performanceproblemen.

So wurde letztendlich eine 3-Tier Architektur mit einem Java Client und einem IBM Mainframe APL2 Rechenkern gewählt, die von einer Integrationskomponente verbunden werden. Als Protokolle werden Corba und XML eingesetzt.

2004/2006: Downsizing Projekt auf IBM Workstation APL2

Mit der Vorstellung der Java Schnittstelle ergaben sich neue Möglichkeiten. Da gleichzeitig auch ein Kostendruck auf dem Mainframe bestand, wurde beschlossen, eine Migration auf IBM Workstation APL2 durchzuführen.

2006 - : Die Zukunft von APL bei Thomas Cook

Nach der Sanierungsphase wird Thomas Cook in den nächsten Jahren wieder angreifen.

Die Thomas Cook AG und IBM Deutschland haben im Juni 2006 den Vertrag über die Entwicklung einer neuen IT-Plattform im Rahmen des Projekts GLOBE unterzeichnet.

Das Projekt GLOBE der Thomas Cook AG steht für eine völlig neue, internationale IT-Strategie. Ziel ist die Entwicklung einer einheitlichen Pro-

duktionsplattform für alle Veranstaltermarken der Thomas Cook AG. Das Unternehmen trägt damit dem veränderten Kundenverhalten Rechnung und gestaltet die Produktion effizienter, flexibler und schlanker. Mit GLOBE wird es möglich, jedes Reisepaket erst zum Zeitpunkt der Kundenanfrage zusammenzustellen und individuell zu bepreisen („Dynamic Packaging“ bzw. „Dynamic Pricing“).

Die neue IT-Plattform setzt konsequent die Idee der Service orientierten Architektur (SOA) um. Dank seiner Integration in die IBM Websphere Produktfamilie ist IBM Workstation APL2 voll kompatibel mit der neuen IT-Plattform von Thomas Cook.

Teile der existierenden Kalkulationssysteme lassen sich damit schnell zu Services umbauen und in andere Anwendungen integrieren.

Damit sind für APL die Voraussetzungen geschaffen, auch zukünftig an den Entwicklungen von touristischen IT-Systemen teilzuhaben.

Autor:
Uwe Schwagmeier
uwe.schwagmeier@thomascookag.com

Betriebswirtschaftliches Informations- und Analysesystem STS.win

Fast 30 Jahre APL im Rheinischen Sparkassen- und Giroverband, Düsseldorf

Der Rheinische Sparkassen- und Giroverband in Düsseldorf (RSGV) ist die Dachorganisation für *35 Sparkassen im Rheinland*. Die rheinischen Sparkassen hatten zum Jahresende 2005 ein Bilanzvolumen von 152,8 Mrd. Euro und beschäftigten 34.806 Mitarbeiter.

Bereits *im Jahr 1977* hat der RSGV damit begonnen, statistische Information seiner Sparkassen in einem selbst entwickelten dreidimensionalen Datenbanksystem (OLAP) unter IBM APL zu verwalten und auszuwerten (Sparkassen-Time-Sharing STS). APL hat mit seinen n-dimensionalen Datenstrukturen schon immer OLAP-Anwendungen hervorragend unterstützt.

Heute werden die statistischen und betriebs-/volkswirtschaftlichen Daten des RSGV in einer Client-Server-Anwendung mit einer modernen Windows-Oberfläche unter IBM-APL2 verwaltet und für Auswertungen bereitgestellt (**STS.win**). Die umfangreichen Datenbanken liegen auf dem Großrechner des Sparkassenrechenzentrums und werden über TCP/IP (Cross-System Shared Variables) gepflegt und abgefragt. Die wichtigsten Bestandteile von STS.win sind:

- *STS.win Datenbanken* liegen auf dem Großrechner, auf Servern der einzelnen Institute oder auf privaten Laufwerken. Sie können bis zu neun Dimensionen haben und werden mit dem benutzerfreundlichen Verwaltungsprogramm *TRADA* aufgebaut und verwaltet. Die zentralen Großrechner-Datenbanken, auf die alle STS.win-Anwender zugreifen können, bestehen zur Zeit aus 45 Datenwürfeln mit ca. 560.000 Zeitreihen.
- Mit dem *Auswertungs- und Analysesystem AIDA* können unter Berücksichtigung von Zugriffsberechtigungen die OLAP-Datenwürfel auf unterschiedlichste Art und Weise (Sparkassenvergleiche, Zeitvergleiche etc.) *individuell* abgefragt und um zusätzlichen Funktionen (Summen, Mittelwerte etc.) ergänzt werden. Zur individuellen Weiterbearbeitung können alle Auswertungen ins Excel übertragen werden.

Außerdem stehen ca. 900 vorbereitete *Standardberichte* mit einer individuellen Auswahl der Sparkassen und der Zeitachse zur Verfügung. Die meisten Berichte werden nicht mehr programmiert, sondern über *Programmgeneratoren* erzeugt, welche die textlichen Vorgaben direkt in APL2-Programme umsetzen.

- Mit dem *Datenerfassungs- und -Prüfprogramm SPKPOOL* erfassen und pflegen die Sparkassen ihre eigenen, für den zentralen Datenpool bestimmten Daten.
- Mit dem *Report-Generator* können mit minimalen APL-Kenntnissen anspruchsvolle Berichtssysteme auch individuell entwickelt werden.
- Eine Vielzahl von *Schnittstellenprogrammen* ermöglichen den Datenaustausch zwischen unterschiedlichsten Datenlieferanten. *APL ist auch hier einfach stark.*

STS.win wurde vom RSGV selbst entwickelt und basiert zu 100% auf IBM APL2. Das RSGV-Team für Entwicklung und Anwendungsbetreuung des STS.win besteht aus je zwei Mitarbeitern für die Bereiche System- und Anwendungsentwicklung. *STS.win wird z. Zt. von ca. 400 Mitarbeitern der rheinischen Sparkassen und im RSGV individuell genutzt.*

Autor:
Klaus-Peter.Friedrich
Klaus-Peter.Friedrich@rsgv.de

APL2 bei DaimlerChrysler

Workstation APL2 V2 wird bei DaimlerChrysler sehr erfolgreich auf Turbolader-Prüfständen in quasi-Echtzeit verwendet, und zwar zur Unterstützung und Auswertung der Versuche mit umfangreichen grafischen Darstellungen und zur Analyse der Daten von Kennfeldmessungen an Turboladern unterschiedlichster Größe und Ausführung.

Versuchsunterstützung

Auf dem Prüfstand werden die Messkanäle der APL2-Anwendung zugeordnet. Nach Beginn des Turbolader-Versuchs sucht eine Funktion ständig nach neuen Messpunkten, übernimmt dann automatisch die Dekodierung der binären Messdaten, berechnet laufend die Kenngrößen und stellt sie sofort nach der Messung grafisch in 4 Diagrammen dar. Optional sind Kennfelder von vergleichbaren Turboladern in den Hintergrund ladbar. Die Maßstäbe der Diagramme werden automatisch an die neu hinzukommenden Messpunkte angepasst. Dies erlaubt dem Prüfstandspersonal die sofortige Überprüfung der Messqualität und Plausibilität.

Auswertung

Hierzu werden sämtliche thermodynamische Kenngrößen berechnet und durch Interpolation durchgezogene Kurvenverläufe erzeugt und grafisch dargestellt (z. T. mit Höhenlinien und farbkodiert). Die Daten werden in APL2-Strukturvariablen sowohl im Arbeitsbereich als auch parallel in einer AP211-Datei verwaltet. Durch Überlagerung von beliebig vielen Kennfeldern kann die Charakteristik von verschiedenen Turboladern miteinander verglichen werden. Interpolationsverfahren mit Bezier-Splines und Berechnung der zugehörigen Kontrollpunkte bilden die Grundlage für eine mögliche Erweiterung, den Interpolationsverlauf durch Ziehen mit der Maus korrigieren zu können. Die Anwendung kann die tabellarischen Auswertungsdaten nach Excel (auch mit den Interpolations-Koordinaten) exportieren sowie grafische Darstellungen nach Powerpoint. Fremdgemessene Kennfelder aus Excel-Tabellen können auch in die Anwendungsdarstellungen importiert werden.

Analyse

Bei unplausiblen Daten können Ergebnisse, Zwischenwerte und Rohdaten so in einem gemeinsamen Diagramm dargestellt werden, dass die gegenseitigen Abhängigkeiten erkennbar sind und auf Fehler rückgeschlossen werden kann. Die Daten erhalten für die unterschiedlichen Typen (oder Typgruppen) jeweils eine eigene Y-Achse, deren Maßstab automatisch (möglichst formatfüllend und passend zum gemeinsamen Gitter) bestimmt wird. Ein „Strichcursor“, (eine bewegliche senkrechte Linie), der an den X-Koordinaten der Messpunkte einrastet, liefert die digitalen Tabellenwerte aller zu diesem Punkt gehörenden Daten. Ein zweiter Strichcursor liest ebenfalls die Tabellenwerte an seiner Position aus und zeigt diese als Differenzen zu den Tabellenwerten des ersten Strichcursors an. Diese beiden Strichcursor dienen dem Prüfingenieur zur genaueren Beurteilung des Verhältnisses von Messpunkten zueinander. Bei Zweifeln an der korrekten Funktion des Messkanals oder an der Stabilität des Messpunktes können sehr einfach die Inhalte der binären Messdateien statistisch analysiert und grafisch dargestellt werden (z.B. deren Verlauf über der Messzeit).

Fazit

Workstation APL2 V2 hat sich bei der Entwicklung und im Betrieb des besagten vielfältigen Anwendungspakets seit Jahren sehr gut bewährt.

Autor:
Winfried Sommerer
Winfried.Sommerer@gmx.de

The Sandvik CAPP system

We have increased productivity by 30%

In the summer of 1984 a project focusing on computer-aided process planning (CAPP) was started at Sandvik to rationalize design and process planning.

The purpose of the CAPP system is to simplify and speed up the preparation of offers and orders for special products.

The objective was to reduce the quotation time from 3-4 weeks to 24 hours and the delivery time, which in many cases could amount to over 10 weeks, to 1-2 weeks, for each order of a special product. A quotation of a special product could need the skills of engineers and marketing people.

Today CAPP covers more processes such as order handling, inventory control, production planning, pricing of special products and standardization of production methods. It provides answers to queries concerning such areas as delivery time, production cost and recommended base price. With CAPP an offer can be prepared immediately when all prompts have been answered and the salesperson doesn't need the knowledge of how to design, construct or price the product.

When the salesperson makes an order, the information necessary for the production of the order will immediately be sent to the plant that is selected for production.

Some products also have CAD support and for such products, data is sent to a CAD system that will immediately prepare production drawings to scale, NC-programs etc. which will also directly be sent to the selected plant.

Advanced products that earlier took weeks to produce now requires only a couple of hours.

There are about 1,500 users throughout the Sandvik organization, most of whom are internal salespersons and distributors, while others work in pre-production or design. However, there is another group of users who are trained developers. In CAPP, they have their own programming language, in which they can create programs based on their operational knowledge and production logic.

The tool behind the language was created and still maintained by the APL2 development group within Sandvik. This group contains of 5 persons working full time with APL2.

The whole system is built in APL2 under TSO in mainframe z/OS. It is used worldwide but only one version of the system exists in the central mainframe.

The CAPP coordinator at Sandvik Coromant, handles contacts with users in the sales department. He says „They turn to me with questions about tenders and tell me how they wish to work with CAPP. I enjoy excellent cooperation with the guys at the CAPP development team, to whom I pass on technical questions about systems. CAPP is extremely stable and offers short waiting times. It hardly ever freezes and we have not had any performance problems."

Rune Karlsson, Vice President Tooling Supply, says, „We have increased productivity by 30%."

Author:
Rune Karlsson, Vice President Tooling Supply, The Sandvik CAPP system, Sweden

University of Puerto Rico

Colleagues express awe when I share my results

My experience with APL2 includes classroom and research activities. APL2 is available to faculty through the IBM Academic Initiative Program for such activities. Presented below is a gear related scenario on how APL2 was and is currently being used as a research and development tool.

As part of my research, a system of coordinates, geometric relations, and a fabrication process were proposed. The gearing industry is a „hands-on" community that demands proof of concept to warrant serious consideration of an alternative technology. The many mathematical relations developed are computationally intensive and need to be translated into computer code and tested. APL2 was selected over Fortran, C, MathCad, and Matlab to develop and demonstrate the functionality of these mathematical relations. Colleagues express awe when I share my results and the amount of computer code used to obtain these results. Moreover, I consider myself a „non-programmer" within the software community, yet APL2 allows me to write the necessary software to „quickly" obtain results. Examples include polynomial root solving, eigenvalue/eigenvector, FFT (Fast Fourier Transform), and matrix manipulation routines used in noise and vibration prediction or FEM (Finite Stress Analysis) stress analysis of gear elements. I estimate that time and effort to write this same code using another computer language to double or triple.

The growth of APL2 and its support of GUI (Graphical User Interface) programming has further allowed me to continue with APL2. In order to demonstrate the benefits of the developed mathematical relations and supporting computer code, a GUI package is being developed such that engineers within the gearing community can implement this technology without being burdened with mathematical relations. This GUI interfaces with the APL2 graphics package GRAPHPAK to display line graphs, bar charts, pie charts, contour plots, and surface plots. APL2 also provides support to output data in Excel files for use by other software packages. This GUI along with all supporting software is implemented using a single environment, APL2. It is my impression based on communication with fellow engineers that this same GUI package would require an „experienced" programmer using another programming language. I am able to distribute this GUI using the APL Runtime Modules where gear designers can use this software without purchasing APL2. Moreover, APL2 automatically generates the GUI using Unicode.

APL2 has allowed me to remain focused on a new gear technology without getting distracted with software development. I feel that much of what I have proposed and developed would not have emerged as such without the availability of APL2. I continue to use APL2 in the „what-if" or „show-me" scenario and recommend for all results oriented people. I have been a user of APL2 for the past 15 years and gauge that the language continues to evolve. I look forward to each release as well as participating in its growth.

Author: David Dooner
Customer: University of Puerto Rico

A strategic investment returning immediate pay-back

Landstinget Vastmanland is a Regional Medical Service Center in Sweden with about 6,000 employees, most of them doctors and nurses.

Olle Berg, Senior IT Manager between 1985-1996, states,

„In 1993 we had a difficult IT situation with a rapidly growing backlog of user requirements within a wide range of areas and often a demand for almost immediate delivery. Our IT organisation was not at all prepared to meet these expectations and I personally investigated a number of ways how to solve this issue. To hire lot of external consultants for traditional mainframe or PC-based development or to buy a set of external application packages were less promising for cost and flexibility reasons as well as risk exposure. After careful consideration my final decision was to introduce APL2 as our premier development environment. This decision was taken after a successful pilot installation and estimations of what we could expect to achieve. One of the major issues we faced was how to manage role-based access to information stored in central databases, without the need for complex products or costly administration. Requirements to secure very good performance for all SQL queries executed was almost immediately solved in the APL2 environment which made our confidence grew and the decision was finally easy to take. We dedicated a small team of experienced developers to provide the services, covering all areas from development to operations and support. The results soon exceeded by far our high expectations and later estimations have shown that we saved a number of million USD over the next few years. I also believe that we could not have been able at all to solve all tasks without APL2. The speed, flexibility to change, overall cost and quality provided during those years with APL2 are still unmatched."

Olle Berg continues:

„Within a period of three years, development and implementation of a number of applications took place, with hundreds of users introduced within a few weeks after deployment. High development speed and a very reliable operations environment were from the beginning critical success factors. Among the applications we introduced was a data query and reporting tool which was used by many users, either for free ad-hoc queries or prepared reports. All users were using this under the control of the security system we designed to provide role-based access to information in databases and files. I also remember when a critical lack of functionality was discovered in our payroll system. We managed to overlap this gap by quickly developing and testing an APL2 solution for this critical need, just in time to avoid serious delays in the payroll process.

Almost all work was done by a small development team with one senior developer and two part-time senior consultants, assisted by a system programmer who provided technical services i.e. DBA and backup services. The development work was mainly done in close co-operation with business representatives using agile development methods similar to those found today in Extreme Programming (XP) and other modern methods.

My personal opinion is that the total savings achieved by this successful APL2 development, compared to other available alternatives, exceeded ten million USD seen over the first six-year period. The savings in speed and agility are not easy to quantify but all applications were developed, tested and put into production within a few weeks or months including pre-study work. We have neither before nor after seen something comparable"

Bengt-Ola Isetoft, former IBM employee and during 1993-2000 responsible for the APL2 development team at the Regional Medical Service comments:

„IBM's APL2 implementation in mainframe was, and still is, from my point of view:

- A full-ledged and mature environment for development and execution of a wide range of applications
- One of the most cost-efficient environment for more complex application development, seen over the total application lifecycle
- Overall the most stimulating and exciting development environment I have ever seen. Today when everybody talks about SOA, Service-Oriented Architecture, I realise that I have been working according to these principles since the early 1980'ies thanks to APL2.

We based our core development on APL2 mainframe, in a small-scale VM/SP environment which existed during the years 1992 - 2000 beside legacy systems in a CICS/VSE environment. At the same time a number of PC applications and groupware were introduced to provide office and collaboration features to thousands of users. APL2 was the base in a strategy to establish in-house systems development services to provide new IT solutions to many users and solve urgent needs within a very short timeframe. The applications were initially Data Warehouse and statistical systems, based on a common report- and analysis tool and environment but there were also a number of complex applications developed which solved needs in a number of areas. APL2 was also used as a converter and bridge between systems which had none or immature interfaces for data exchange.

The main benefits by the APL2 development were:

- Very fast application development and maintenance
- Heavy user involvment depending on

the highly interactive environment, which lead to high-quality and well tested applications

- Prototyping and traditional development was almost the same, the quality of prototypes was good enough to put into production since they were developed with the same methods and same technology. What differed between a prototype for evaluation and a ready-for-production application was often just the amount of testing. - applications which could be enhanced and improved over many years without decreasing flexibility or quality, depending on design and a service-oriented approach, SOA before it was commonly known and accepted.
- Very low total cost, both for development, operations and overall lifetime

I wish for the future that the APL language continues to live and develop and I am still totally convinced that the unique design of APL is the best implementation of a developer-friendly computer language we have ever seen“

Customer: Landstinget Vastmanland
Authors: Olle Berg and Bengt-Ola Isetoft, Sweden

Software that „Feels Right“ to IBM researcher

Personally, I consider APL2 the most technically challenging software produced by IBM - ever. Of course, I saw quite a few complex and massive software projects - and I fully appreciate how difficult it is to put together something like MVS or Notes; however, I can imagine how a well funded group of reasonably smart people could make it happen. To produce APL2, however, one would need to have some super-smart people on board... Somehow, this software just „feels right“.

One of the most important aspects of APL2 is its multi-platform nature. This enables me to write industry-grade applications for use in a mainframe or multi-processor Unix environment by using my preferred Windows development environment, where I can use APL2 in conjunction with so many „goodies“ provided by Microsoft and its followers. It is in this environment that I keep my testing setups, logbooks, notes and papers. Once the development is complete, I simply FTP the workspace to a production Unix machine and re-run the tests... just in case (usually, no modifications are needed because I use a set of functions that „adapt“ to the operating environment). Then I simply make the workspace available to a production system. This is how APL2 is currently running a massive monitoring program for detection of unfavorable changes in our semiconductor manufacturing operation. This system is in a 24/7 operation for about 1/2 year - and so far, we had 100% availability and a very positive feedback from the users. Most of them do not even know that APL2 is involved in the process.

Another recent application (Early Detection Tool) was developed for massive monitoring of IBM Pc Co warranty data. This application was also developed and tested on Windows, but then it was deployed in a mainframe environment prevalent in this division (now Lenovo). The main challenge there was to make sure that the system can explore about 50000 various combinations of machine types and components so as to detect whether some combinations should be brought to the attention of Quality, Procurement or Brands people. The challenging point in such a system is to assure not only good detection capability, but also a low rate of false alarms - and this, in turn, requires some advanced statistical algorithms (these were also developing under APL2). This work led to Research Division Accomplishment in 2004. It is interesting to note that the application was originally developed as a prototype, and it was anticipated that in a subsequent stage this prototype would be re-written into C. However, when the PC Co. people saw that their worldwide warranty data could be fully and reliably processed within 36 hours by just three parallel accounts on the mainframe - they decided to forgo the re-writing and simply develop the system as an APL2 solution.

With best regards,
Emmanuel

Customer: IBM Research
Author: Emmanuel Yashchin
Country: United States of America

Real Time Processing at the Missile Defense Agency

I worked in the field of ballistic missile surveillance from 1969 until 2003. Throughout this time, I first used APL, and then APL2, to prototype tracking and estimation algorithms that I then translated into either Fortran or Ada. Some of this work is described in my APL93 paper, „Roles of APL in Satellite Surveillance" and in the paper, „Surveillance and Tracking of Ballistic Missile Launches" in the March 1994 issue of the IBM Journal of Research and Development.

I still remember my chagrin when President Nixon signed the ABM treaty with the USSR in 1972. For some time I had been studying Soviet tests of extremely threatening strategic systems, and I loathed our official policy of Mutual Assured Destruction. My goal was not to watch hostile missiles helplessly, but instead to have the capability to destroy them before they could reach American soil.

In 2002, at long last, President Bush dissolved the ABM treaty and Congress authorized contracts to begin missile defense work in earnest. Existing programs were transitioned from research to development and deployment. Many readers will be familiar with the Ground Midcourse Defense (GMD) system that recently intercepted a target missile that mimicked a North Korean Taepo-Dong ICBM. There have also been successful intercepts of target missiles by the Navy's Standard Missile 3, and by the Army's Theater High Altitude Area Defense (THAAD) missiles. All of these systems intercept the target missile in its midcourse or terminal phase, not its boost phase.

There also arose two new development programs that are designed to destroy a hostile missile in its boost phase, before or soon after it burns out, and before it deploys decoys or other countermeasures. One of these is the AirBorne Laser (ABL).

The other is the Kinetic Energy Intercept (KEI) program, which is a hit-to-kill system like GMD, THAAD and SM-3. The main differences are that KEI has much less time to react, it needs more powerful interceptors, and it needs more sophisticated (and very fast) tracking and prediction algorithms.

My work involves the „prediction" portion of the KEI program. The objective here is to predict the future trajectory of the target missile so that an interceptor missile can be directed to the right time and place for a successful intercept. This prediction is first made shortly after the launch of the hostile missile has been detected. The prediction is then continually refined based on additional surveillance sensor inputs, so as to provide real-time directives to the interceptor missile to maneuver as necessary until its own sensors can take over.

As usual, I have used APL2 to prototype all of my work on this program, including performance testing against both real and simulated inputs. Those prediction algorithms that prove their worth are then migrated to C++ and integrated with the tracking algorithms.

Attached is General Henry Obering's letter of commendation to the CEO of Northrop Grumman regarding our team's recent work on KEI. General Obering is head of the Missile Defense Agency.

Note particularly his remark about algorithm performance. This refers to the „Tracking and Prediction" algorithms that our Boulder team developed and implemented in a real time processing system. These algorithms use sensor data in real time to track and predict the future trajectory of a threatening ICBM so as to direct KEI's interceptor to hit it before, or shortly after, it burns out (and before it can deploy decoys).

Our Prediction algorithms were first prototyped in recent years (by yours truly) in APL2. Also some of the tracking algorithms were originally prototyped in APL2. Therefore, our prototyping work using APL2 is largely responsible for Obering's remark about our ability to direct the interceptor toward where the target will be at the time of interception.

And therefore, you great folks in IBM APL Products and Services have been helping America's defense against missile attack, whether you have known it or not.

Thank you very much for all you've done over the decades.

Sincerely, Jack Rudd

P.S. One related (and unclassified) piece of work may surprise even experienced users. I downloaded from NOAA's website a global topographical map that was partitioned as 16 large files. I then used APL2 to create from this information a single file that contains a single byte for each of 21600 latitudes and 43200 longitudes all over the globe. Each byte contains the elevation above sea level for the corresponding position on the globe, with a vertical resolution of 35 meters. (Thus covering the range from sea level to the peak of Mount Everest.)

This file contains nearly one gigabyte of data, and it is used as a simple lookup table. Given any latitude and longitude, to a resolution of 1/120 of a degree, one locates and reads a single byte from this file to obtain the corresponding elevation above sea level. Statistically it turns out that APL2's Auxiliary Processor 210 takes no more time than C to obtain this value (at least under Windows).

Thus neither workspace memory nor cycle speed nor auxiliary storage access time was a limiting factor for using APL2 for this rather large problem. This will likely come as a surprise to many, and I hope it helps counter some ancient myths about APL performance.

Customer: Northrop Grumman
Author: Jack Rudd
United States of America

Allgemeine Informationen
(Stand Mai 2005)

APL-Germany e.V. ist ein gemeinnütziger Verein mit Sitz in Düsseldorf. Sein Zweck ist es, die Programmiersprache APL, sowie die Verbreitung des Verständnisses der Mensch-Maschine Kommunikation zu fördern. Für Interessenten, die zum Gedankenaustausch den Kontakt zu anderen APL-Benutzern suchen, sowie für solche, die sich aktiv an der Weiterverbreitung der Sprache APL beteiligen wollen, bietet APL-Germany den adäquaten organisatorischen Rahmen.

Auf Antrag, über den der Vorstand entscheidet, kann jede natürliche oder juristische Person Mitglied werden. Organe des Vereins sind die mindestens einmal jährlich stattfindende Mitgliederversammlung sowie der jeweils auf zwei Jahre gewählte Vorstand.

Vorstand

1. Vorsitzender

Dr. Reiner Nussbaum
Dr. Nussbaum gift mbH
Buchenerstrasse 78
69259 Mannheim
Tel. (0621) 7152190

Schriftführer/Internetbeauftragter

Michael Baas
Dynamic Logistics Systems GmbH
Wilhelm-Schöffer-Str. 29
63571 Gelnhausen
Tel: (06051) 13068
Fax: (06051) 16142
eMail: m.baas@dls-planung.de

2. Vorsitzender

Martin Barghoorn
Sekr. FR 6-9
Technische Universität Berlin
Franklinstr. 28
10587 Berlin
Tel. (030) 314 24392

Schatzmeister

Jürgen Beckmann
Feudenheimer Grün 10
68259 Mannheim
Tel: 0621-7980840
eMail: JBecki@onlinehome.de

Beitragssätze

Ordentliche Mitglieder:

Natürliche Personen	32,00	Euro*
Studenten / Schüler	11,00	Euro*

.Außerordentliche Mitglieder:

Juristische / natürliche Personen	500,00	Euro*

* *Jahresbeitrag*

Bankverbindung:

BVB Volksbank eG Bad Vilbel
BLZ 518 613 25, Konto-Nr. 523 2694

Hinweis:

Wir bitten alle Mitglieder, uns *Adressenänderungen* und *neue Bankverbindungen* immer sofort mitzuteilen. Geben Sie bei *Überweisungen* den *Namen* und/oder die *Mitgliedsnummer* an.

Einzugsermächtigung

Mitglieds-Nr.: _____

Ich erkläre mich hiermit widerruflich damit einverstanden, daß APL Germany e.V.

den jeweils gültigen Jahres-Mitgliedsbeitrag von meinem unten angegebenen Konto abbucht.

Einen eventuell bestehenden Dauerauftrag habe ich bei meiner Bank gelöscht.

Bankbezeichnung: _____

BLZ: _____ Konto-Nr.: _____

Datum: _____ Unterschrift: _____