Journal

HP General Systems Users Group

WINTER 1980

Executive Office
HP General Systems Users Group
Empire Towers
7300 Ritchie Hwy.
Glen Burnie, MD 21061
USA

Rella M. Hines, Executive Director

SPOTLIGHT ARTICLES
Lyon Proceedings
Interactive Project Analysis and Control System 3
Jerry Smithers

Design and Implementation of REX/3000 4
Lance Carnes

Dollar-Flow: HP3000 Financial Planning 9
Jack Damm

FEATURE ARTICLE 14
FORTRAN/3000 and FORTRAN 77: A Comparison
James P. Schwar and Charles L. Best

LIBRARY CORNER 16
Software Exchange at San Jose
Wayne Holt

TIPS & TECHNIQUES 17
Hints on IOSTAT2
Robert M. Green

RETURN CARD 23
Bug/Enhancement Poll
Ross Scroggs

THE JOURNAL NEEDS YOU!

The **JOURNAL** *of the HP General Systems Users Group depends on contributed articles to fulfill its purpose of disseminating information to members of the organization. Current articles are uniformly of high quality and provide excellent information for the membership. However, the Executive Board and the Publications Committee would like to expand these offerings.*

This request is addressed to those readers with ideas to be shared. The Publications Committee encourages all readers to contribute the results of their work. The **JOURNAL** *offers an excellent vehicle for publication of current work for the established computer person as well as for the beginning user. Articles of all types are needed. As installations increase the need to know, to share ideas, and to establish written communications among members of the Users Group increases at an even faster rate.*

Remember, the areas where problems and failures have a high probability of occurring are of interest to a large number of other HP users. Share information on your successes and indicate the future direction of your work by writing articles for the **JOURNAL**.

The 1980 **JOURNAL** Publication Schedule, as given below, includes the deadlines for receipt in the Executive Office of articles submitted for publication in each issue. Also given, is the issue in which the article will be published and the month in which HPGSUG members will receive each issue. Please note that additional time has been allowed for any delays caused during peak mail periods. If you do not receive your **JOURNAL** during the time allowed, please notify the Executive Office at (301) 768-4187.

1980 HPGSUG **JOURNAL** PUBLICATION SCHEDULE

Articles Submitted by:	Issue(s)	Members Receive by:
November 16	Winter	January 1980
February 11	Spring	April 1980
April 18	Summer	June 1980
July 18	Fall	September 1980
October 17	Winter	January 1981

Contributions: HPGSUG Executive Offices
ATTN: Carol W. Taylor
Publications Manager

SPOTLIGHT ARTICLE

Interactive Project Analysis and Control System

Jeremy Smithers
London Business School
London, England

BACKGROUND

Network Analysis is one of the oldest computer applications particularly in the management science field. Over the past twenty five years a large number of packages have been developed for large mainframe computers. These packages were extensively modified to include new facilities as the technique grew more sophisticated. As a result of this project engineers either had to make a considerable investment in understanding how to use these systems or be assisted by a specially trained operator. These systems were also designed for batch processing and although interactive front-ends were added, they were mainly simple file editors.

With the advent of large scale time-sharing systems new packages were developed with specific interactive capabilities. However, it has proved difficult to control the cost of project management using these systems. However, the same benefits may be achieved by using a suitable package on an in-house mini-computer with the advantage of being able to control the cost.

The major requirements for such a package is that it should be easily usable directly by the project engineers, at the same time providing enough facilities to realistically simulate the project environment.

INTRODUCTION

Interactive Project Analysis and Control System is an on-line package for planning and control of a project written in FORTRAN and designed to operate in a mini-computer time-shared environment.

It is derived from a package called ACSPERT which was developed in 1971 to provide interactive PERT/CPA facilities via a time-sharing bureau. This package has been in operation since 1972 and has been extensively upgraded to meet the requirements of a number of large users. Consequently IPACS derives the benefit of considerable experience of interactive network analysis.

ACSPERT main features are ease of use and efficiency of operation. These features have been retained in IPACS but considerable enhancements have been made to the project control area. IPACS provides sufficiently detailed facilities to allow it to control individual resources to the nearest manhour.

IPACS has been developed in a modular open-ended format. This means that new facilities and extra options in existing facilities may be added with the minimum disruption. The file layouts have been designed to record most detailed information but the program is written so that extra information may be easily stored.

There are no absolute limits on the size of network that may be analyzed, this being dependant on the amount of memory available. On a 64K byte mini-computer with a time-shared operating system it is designed to handle networks between 3000 - 4000 activities.

FEATURES

* Free format input with default values for all items
* Activities with multiple (complex) resources
* Imposed milestones
* Holidays
* Responsibility codes and costs
* Resource man-hours/week and costs
* Interactive analysis of dates
* Sub-networks
* Hammock activities for overheads
* The Report generator has control facilities
  - to select, group and sort activities
  - to specified special headings and non-standard time periods
* The Report generator produces the following planning reports:
  - Time Analysis
  - Bar Chart
  - Resource Analysis (tabular/histogram)
  - Cost Analysis
  - Event/Milestone Report

CONTROL FEATURES

* Both the original plan (budget) and the actual implementation may be stored;
* The Field Progress report shows the actual and expected progress over the reporting period;
* Only exceptions from the forecast progress need be input via the progress commands to up-date the project;
* The management cost report shows actual against budget progress over the reporting period
* Resource and cost reports show actual and projected actual against budget.

SYSTEM OVERVIEW

IPACS provides a system for maintaining a database on a project. The database is contained in two separate files. Firstly the Project Network Details (PND) which contains an entry for each activity. Secondly the Project Reference details which record all other information (e.g. titles, milestones, holidays, resources, responsibility codes, costings, etc.).

A new project is initially set up by using the NEW command. This will ask for some reference details and a specification of each activity. After a new project is established HOLIDAYS and MILESTONES may be entered and RESOURCE and RESPONSIBILITY CODE profiles defined. These commands also provide facilities for editing and listing all these details.

The network data may be subsequently EDITED where there are a set of facilities for adding/overwriting activities, modifying individual fields and listing selected activities based on criteria about individual fields. Activities may also be entered as SUB-NETWORKS from a library. This means that standard tasks need not be redefined each time they occur.

A project may be ANALYZED to establish the earliest and latest dates for each activity taking into consideration imposed milestones and holidays. All activities are considered to start at this earliest date unless they are SLIPPED back.

Once a time schedule has been established for a project then a series of reports may be produced. These show the time schedule of the project in clear tabular form (TIME ANALYSIS) or in graphical form (BAR CHART). The resource requirements (RESANAL) and costs (COSTANAL) can be shown in tabular or histogram form. An EVENT (or Milestone) report shows key dates. Each report may be controlled as to which activities should be analyzed, the sequence and grouping of the activities, the dates covered, and special formatting requirements.

The initial analysis of the project is the planning and evaluation stage. Once a plan for the project has been finalized then the updating mode is changed to ACTUAL. This preserves the initial plan as a budget against which to measure the actual progress of the project.

Updating and monitoring the project is done on a reporting cycle. Firstly a Field Progress Report is produced showing for all current activities the actual and forecast progress up to the reporting date. This report is used on an exception basis, only differences from the forecast need be recorded by the field supervisors. These differences are entered via the PROGRESS command which automatically progresses all other activities in line with the forecast. At the same time as the project is progressed, a Management Cost Report is produced showing the actual progress compared with the budget. At this time the next FPR can be produced which starts the next loop of the reporting cycle.

Design and Implementation of REX/3000

A Natural Language Report Writer

Lance Carnes
Gentry, Inc.
Kensington, CA

This paper presents a natural language report writer, REX/3000 (Report Expediter), designed for the casual user as well as the experienced programmer. The language design and compiler implementation is discussed from the point of view of meeting the needs of the two user classes. For the casual user, the language is natural and English-like. For the programmer, the language is a mixture of natural and procedural constructs. The PASCAL-based compiler has been implemented to serve each user class appropriately.

INTRODUCTION

REX was developed to facilitate writing reports from IMAGE databases and MPE files. The user will range from the casual (infrequent) non-programmer attempting to formulate reports to the experienced programmer with more complex goals.

The time allotted for the design and implementation was eight man-months. The project consisted of the following:

1) Design the syntax of REX, i.e. the form of the language.

2) Design the semantics of REX, i.e. the function of the compiled programs, to properly execute for both the casual user and the programmer.

3) Write a compiler for REX.

This seemingly tight schedule was supported by several resources:

1) The author of a report writer similar to the required package (though unfortunately written in IBM 360 assembler language [1]) was available for consultation;

2) A PASCAL compiler had been transported to the HP by a user, and graciously contributed to the user's library [2];

3) A copy of the production PASCAL-P compiler source (written in PASCAL) had also been contributed to the users library.

This paper will treat these topics, discussing the design considerations, and the solutions implemented.

(REX/3000 continued)

SYNTAX

The careful design of the syntax of a language is important. From the compiler writer's point of view, the syntax should be simple and have as few rules and exceptions as possible. From the language user's point of view, especially the casual user, the syntax should be natural and English-like as well as consistent and unambiguous.

The syntax of a language is the way the elements of the language are put together to form statements. For example, in English many statements have the syntax

    subject verb object.

In computer languages the syntax of assignment statements are commonly:

    variable = number

or variable = number + number

or, in the general case,

    variable = expression

The syntax of REX lies somewhere between a typical programming language and a natural language. The total logical precision of a programming language is not wanted, since the casual user is not trained in the use of these languages. Nor do we want the full complicated syntax of English, since the language must be translated by a computer program (the compiler).

The syntax of REX is modeled after the syntax of many of the currently used "natural language" report writers: SYNTAX II [1], QUERY [3], NATURAL [4], R11 [5], and RAMIS II [6]. The intent of all the natural language report writers is to avoid the "procedural" or programming details of programming languages and to allow the user to formulate English-like specifications which can be interpreted by a compiler.

Consider Example 1, a REX program which reads and prints selected values from a database called PAYROLL.

```
<< LIST EMPLOYEES IN SALES DEPARTMENT FROM MARIN COUNTY >>
database PAYROLL password 'READER' access 8
  dataset EMPLOYEES
print 'SALES PERSONNEL RESIDING IN MARIN COUNTY'
get EMPLOYEES with DEPT = 'SALES'
  select ZIP isbetween 94100,94199
  print EMPL-NAME, EMPL-ADDR, EMPL-NUM
end.
```

Example 1. A complete REX program.
(Words in lowercase are keywords - in actual practice upper and lower case letters are treated the same.)

This is a complete REX program and will produce a listing of the desired values. (Its actual function will be discussed in the next section, "Semantics". See Appendix B for sample programs with printed output.)

The point here is that REX has an English-like syntax which serves the casual user. Notice that there are no procedural steps (i.e. there is no explicit opening or closing of the database, no explicit test for end-of-chain, etc). In a sense, the syntax closely resembles the specification of the program.

For the experienced programmer, who must deal with more complex programming requests, there are more complete constructs available. However, these constructs are not always "natural language" in nature and would not be used commonly by the casual user. The constructs are available for completeness and are not in ordinary demand by the non-programmer.

For example, suppose the user wishes to read an MPE file containing employee numbers, look up each employee number in the PAYROLL database, and write a new file with employee name, address and number.

```
<< READ EMPLOYEE #, LOOK UP # IN PAYROLL DATABASE,
<< WRITE NAME, ADDRESS, EMPLOYEE #

file EMP#FILE = 'EMPLNO.PUB.ACCOUNTS' f10  << INPUT FILE >>
  EMP# 1-4 n7  << 4-BYTE FIELD, 7-DIGIT INTEGER >>

database PAYROLL password 'READER' access 8
  dataset EMPLOYEES

file EMP-NAME-FILE = 'EMPLNAMS.PUB' f72 << OUTPUT FILE >>
  E-NAME  1-20 a20   << 20-BYTE ALPHA FIELD >>
  E-ADDR 21-50 a30
  E-NUM  51-54 n7    << 7-DIGIT INTEGER >>

progvar REC-CTR   n8 << 8-DIGIT COUNTER, RECORDS READ >>
        NOT-FOUND n8 << 8-DIGIT COUNTER, INVALID EMPL #'S >>

begin  << MAIN PROGRAM >>

  REC-CTR = REC-CTR + 1  << COUNT INPUT RECORDS >>

  get first EMPLOYEES with EMPL-NUM = EMP#  << LOOK UP >> &
    at end begin
             NOT-FOUND = NOT-FOUND + 1 << LOOK UP FAILED >>
             print 'EMPL # ' EMP# ' NOT IN PAYROLL'
           end
  E-NAME = EMPL-NAME   << MOVE VALUES TO OUTPUT RECORD >>
  E-ADDR = EMPL-ADDR
  E-NUM  = EMPL-NUM
  write EMP-NAME-FILE    << WRITE OUTPUT RECORD >>
loop  << DO NEXT read EMP#FILE >>

<< ALL RECORDS PROCESSED, PRINT TOTALS >>
if NOT-FOUND <> 0 then  &
  print 'TOTAL EMPL #'S NOT FOUND = ' NOT-FOUND  &
else print 'ALL EMPL #'S FOUND
print 'TOTAL RECORDS PROCESSED = ' REC-CTR
end.
```

Example 2. A complex program.

This example illustrates many of the constructs available to the programmer or the adventuresome casual user. See Appendix A for a summary of the REX language.

SEMANTICS

The syntax of a language provides a framework of correctly formatted statements which may then be converted to meaningful, functioning programs. The conversion of statements into functional programs is called the semantics of the programming language. The semantics for REX are such that the constructs used by the casual user require a great deal of implicit function, while the procedural constructs for the experienced programmer are explicit.

For example, in English the request
"Look at your watch and recite the exact hour and minute." may also be stated as
"Tell me the time."

The first form is correct English syntax and there is no difficulty understanding what is desired, although this is not the usual way to ask the time. The second form has the same meaning and can be understood with slightly more effort; i.e. you must subconsciously recall that you need to look at your watch, and recite the hour and minute. The first form is procedural and explicit; the second form is natural and requires a great deal of implicit function.

The semantics of REX are designed to cover both cases - the casual user can express a natural request while the programmer can code the same function procedurally. To illustrate, recall Example 1. Nowhere was it indicated explicitly that the dataset was to be read using a chained read on the DEPT search item, looping back to the "get EMPLOYEES" statement after each entry had been printed. The explicit statement of this program is shown in Example 3.

```
database PAYROLL
  dataset EMPLOYEES
print 'SALES PERSONNEL RESIDING IN MARIN COUNTY'
find EMPLOYEES with DEPT = 'SALES'  << FIND CHAIN HEAD >>
set next EMPLOYEES at end stop  << NEXT ENTRY ON CHAIN >>
  select ZIP isbetween 94100,94199
  print EMPL-NAME, EMPL-ADDR, EMPL-NUM
loop     << END OF set LOOP >>
end.
```

Example 3. Explicit statement of Example 1.

The program in Example 3 will produce the exact same result as the program in Example 1. Details are described which were filled in by the compiler in Example 1. The clause "at end stop" indicates that processing is to cease at the end-of-chain. The "loop" statement indicates that the processing loop ends precisely here, so that statements may be written following the loop which are not executed until the loop is terminated (see Example 2).

In all aspects of REX, the functions which are most used by the casual user have a "natural language" or non-procedural default. Another example of this aspect of REX is the report block. The following program produces a result similar to the program in Example 1 - except here the output is sorted by employee name, and a report title and column headings are provided:

```
database PAYROLL
  dataset EMPLOYEES
report 'SALES PERSONNEL RESIDING IN MARIN COUNTY'
  select ZIP isbetween 94100,94199
  list EMPL-NAME, EMPL-ADDR, EMPL-NUM  &
    sorted by EMPL-NAME
end
get EMPLOYEES with DEPT = 'SALES'
end.
```

Example 4. A sorted report.

Here we have specified the contents of a report (report...end) and the method of obtaining items for the report (get EMPLOYEES...). What is left out of the specification (but included in the implicit function) is the linkage between the "report" and the "get", and the indication of when to sort the data and print the report.

Consider the explicit version of this same program:

```
database PAYROLL
  dataset EMPLOYEES
report MARIN-RES 'SALES PERSONNEL RESIDING IN MARIN COUNTY'
  select ZIP isbetween 94100,94199
  list EMPL-NAME, EMPL-ADDR, EMPL-NUM  &
    sorted by EMPL-NAME
end
get EMPLOYEES with DEPT = 'SALES'
  call MARIN-RES    << UPDATE REPORT DATA >>
loop
print MARIN-RES     << SORT AND PRINT THE REPORT >>
end.
```

Example 5. Explicit statement of Example 4.

Notice that the report block now has an identifier "MARIN-RES" which allows it to be "call"-ed, much as a procedure block. The statement "print MARIN-RES" explicitly specifies the exact point at which the report is output.

Thus the semantics of REX are intended to appeal to both the casual and programmer user. The programmer is not saddled with a restrictive language set and may tackle more complicated programs; the casual user may ignore the procedural aspect of REX or may, with some assistance or experimentation, improve his results by venturing into the procedural constructs of the language. Acceptable results may be achieved from either level of expertise.

THE COMPILER

One of the original goals underlying the implementation of REX was the low-cost development of a compiler which could be transported to other machines. Most of this goal was achieved. The compiler was developed at low cost (eight man-months) and is transportable (written in PASCAL). However, since the object code produced by the compiler is SPL, the package will run only on the HP at present.

A compiler is a program which processes source text in a pre-defined language to produce some form of object code. The syntax and semantics of the language to be compiled can greatly affect the complexity, and therefore the cost, of developing and maintaining a compiler. When aiming at low cost development, it is important to keep it as simple as possible without sacrificing the utility of the language.

REX is not an easy language to compile. However, the task was reduced because a production compiler was used as a model (the PASCAL-P compiler written in PASCAL). Also, some language design decisions eased the translation effort without reducing the capability of the language.

As an example, the element ⟨expression⟩ was used everywhere that it made sense. This allows any production in the language which uses ⟨expression⟩ to share a common compiling procedure. Of course, if the statement needs an expression of a certain type, say logical in the case of "if ⟨expression⟩", the compiler will flag any expression of any other type as an error.

In contrast, the language SYNTAX II [1] has several expression types which are compiled distinctly, depending on the expected type. In the case of the assignment statement SYNTAX II has several different syntactic forms depending on the destination data type.

```
1       A EQ B                           A,B CHARACTER TYPE
2       SET A = B + 1 or A = B + 1       A,B NUMERIC TYPE
3       RECODE A TO B                    A,B ANY SAME TYPE
4       T TEST A < B                       T LOGICAL TYPE
```

Figure 1. Sample assignment statements from SYNTAX II.

The result is that there are several additional compiling procedures that must be developed and maintained.

Using a construct wherever it makes sense is desirable from the compiler writer's point of view and also from the user's point of view [7] [8]. When the user must formulate his idea differently for each data type used, the language designer has burdened the user with additional effort. For example, in the SYNTAX II "SET" statement it is legal to write

```
SET A = B * 100
```

while in the "TEST" statement it is not legal to write

```
T TEST A < (B * 100)
```

even though A and (B * 100) are of the same type. The user must remember which operators can be used in which contexts.

This requires an additional level of expertise that we wish to avoid. However, if there is only one set of rules for the formation of expressions, the user need not recall a lot of exceptions to the rules.

The REX compiler was implemented in PASCAL, a high-level recursive language [9]. The production PASCAL-P compiler source code [10] was used as the basis and as a model for writing the REX compiler. SPL was chosen for the compiler's object code primarily because it is the language closest to the machine level. SPL compiles to produce the most resource-efficient run-time programs. It runs with the least memory usage and fastest execution time compared to other compiled object code (FORTRAN or COBOL). Output in relocatable code was not chosen since there is no documentation currently available from HP on its format.

The appearance of the compiler from the user's standpoint is much like any other HP compiler. There are UDC's for invoking the compiler which resemble the commands for invoking the SPL, COBOL or FORTRAN compilers. Error messages are clear and point to the place in the text where the error occurred:

```
select ZIP isbetwee 94100,94199
**** unknown symbol (GCPERR 72)
```

Should the user want more information, he can refer to the user's manual under "GCPERR 72".

CONCLUSIONS

REX has many powerful features which will allow the user to generate accurate, timely, and complete reports with a minimal program development effort. Results with comparable packages show a decrease in development time by as much as a factor of ten [6], compared with developing the same application in COBOL or FORTRAN.

The casual user can benefit greatly from bypassing the data processing department and writing his own ad hoc reports. The programmer can increase his productivity by reducing the development effort required to produce routine reports and queries.

At the time this paper was written, REX had been implemented with all features except the cross-tabulation, which will be completed by the time this paper is presented. There had been no casual user experience with REX, but by the time of presentation there should be real feedback as to its effectiveness with this user class.

There is no data at present regarding acceptance by experienced programmers. Due to its close resemblance to the "ALGOL-like" languages, it is expected to be successful with programmers with this type of experience. Data will be forthcoming as to its acceptance by programmers experienced in COBOL, FORTRAN, BASIC and other languages.

Future extensions include full IMAGE access (DBPUT, DBDELETE, DBUPDATE), and full KSAM access.

REX will be marketed by:
GENTRY INC.
609 Kearney Street
Kensington, California 94530
U.S.A.

The first release of REX will be available January 1980.

ACKNOWLEDGEMENTS

My thanks to John Fitz, Richard Gentry, and Ron Frankel for many valuable discussions during the

language definition phase of the project; and my special thanks to Grace Gentry, who provided me the opportunity to do this project and who assisted in the preparation of this paper.

REFERENCES.
----

1. John Fitz, SYNTAX II USER'S GUIDE. University of California, 1977.

2. Bob Fraley, PASCALP, HP User's library.

3. QUERY. Hewlett-Packard Product # 32216A.

4. NATURAL Users Manual, Software AG, 1978.

5. RII, Computer Sciences Corp.

6. RAMIS II, Mathematica Corp.

7. G. Weinberg, The Psychology of Computer Programming, Van Nostrand Reinhold, 1971.

8. W.M. McKeeman, "Programming Language Design" from Compiler Construction, F.L.Bauer,ed., Springer-Verlag, 1976.

9. K. Jensen and N. Wirth, PASCAL User Manual and Report, Springer-Verlag, 1974.

10. Urs Ammann, PASCAL P4 compiler source code, Zurich, 1976.

APPENDIX

Appendix A: Summary of REX/3000 Language.

```
REPORT 'TITLE'                          Sorted report.
  LIST ...
    SORTED BY ...
END

TABLE 'TITLE'                           Cross-tabulation.
  ROW ...
  COLUMN ...
END

PROCEDURE name                          Procedure.
  ...
END

DATABASE basename                       IMAGE declaration.
  DATASET setname

FILE filename                           MPE file declaration.
  field1 ...
  field2 ...
  ...

PROGVAR var1 ...                        Variable declaration.
        var2 ...
        ...

                                        Database access.

GET    setname [WITH searchitem = expression]
FIND           [AT END statement]

READ  filename [AT END statement]

WRITE filename                          File access.

PRINT expression [, expression, ...]    Unformatted print.

                                        Sort/Merge.
SORT filename [INTO filename]
          BY key1, key2, ...

MERGE filename, filename [, filename, ...]
          INTO filename
          BY key1, key2, ...

IF expression THEN statement            Control statements.
            [ELSE statement]

GOTO label

CALL blockname

REPEAT statement [; statement; ...] UNTIL expression

WHILE expression DO statement

FOR ident = expression TO expression DO statement


BEGIN [statement; statement; ...] END   Compound statement.

ident = expression                      Assignment statement.


+ - * / DIV MOD                         Arithmetic operators.

GT GE LT LE EQ NE ISBETWEEN CONTAINS    Relational operators.
>  >= <  <= =  <> IB        <*>

AND OR NOT                              Logical operators.

                                        Data types.
Nw     w <= 10          Integer
Fw.d   d <= w <= 15     Real
L                       Logical
Aw     w <= 256         Alpha
```

APPENDIX B: SAMPLE REX/3000 REPORTS.

The following are several reports generated by REX/3000 from MPE files and IMAGE databases.

Sample 1. Sorted report using MPE file.

```
FILE PARTS: F80
   PN 'PART NO' 1-4
   PD 'PART DESC' 5-14
   PL 'LOCATION' 15-17
   QTY 'QUANTITY' 18-22 N5
REPORT 'WAREHOUSE PARTS SUMMARY'
   LIST PN, 5X, PL, 2X, PD, 2X, QTY           &
      SORTED BY PN, PL                        &
      SUMMARIZING QTY ON PN                   &
      TOTALING 'TOTAL' QTY
END
READ PARTS  << READ FILE RECORD >>
END.

1785BOLT 1 X 1/4 101 2000
2142BRACKET      100  750
3122MANUAL #177  101  100         Contents of the
2142BRACKET      102  250         PARTS file.
2142BRACKET      101  100
1785BOLT 1 X 1/4 100 1000
```

The REX/3000 report:

```
               WAREHOUSE PARTS SUMMARY

PART NO      LOCATION    PART DESC        QUANTITY

1785         100         BOLT 1 X 1/4         1000
             101         BOLT 1 X 1/4         2000
-------                                  ---------
1785                                          3000

2142         100         BRACKET               750
             101         BRACKET               100
             102         BRACKET               250
-------                                  ---------
2142                                          1100

3122         101         MANUAL #177           100
-------                                  ---------
3122                                           100
                                         ---------
TOTAL                                         3200
```

Sample 2. Sorted report with IMAGE/3000 database.

The following is an IMAGE/3000 schema for a database to hold the same information from Sample 1:

```
BEGIN DATABASE WAREHOUSE;

ITEMS:
 PART-NO,    X4;
 PART-DESC,  X10;
 PART-LOC,   X4;
 PART-QTY,   I2;

SETS:

 NAME: PARTS, DETAIL;
 ENTRY:
   PART-NO,
   PART-DESC,
   PART-LOC,
   PART-QTY;
 CAPACITY: 100;

...

END.
```

The following REX/3000 report specification will produce the same report as in Sample 1:

```
DATABASE WAREHOUSE
  DATASET PARTS

REPORT 'WAREHOUSE PARTS SUMMARY'
   LIST PART-NO, 5X, PART-LOC, 2X, PART-DESC, 2X, PART-QTY N5   &
      SORTED BY PART-NO, PART-LOC                               &
      SUMMARIZING PART-QTY ON PART-NO                           &
      TOTALING 'TOTAL' QTY
END
GET PARTS  << READ DATASET >>
END.
```

```
Sample 3. Cross-tabulation with IMAGE/3000 database.

This sample uses the same database as in Sample 2.

DATABASE WAREHOUSE
  DATASET PARTS

TABLE 'PARTS DISTRIBUTION'
  ROW PART-NO BINS('1785','2142','3122')
  COLUMN PART-LOC BINS(100,101,102) ACCUMULATE PART-QTY
         TOTAL-PARTS LABEL 'TOTAL PARTS' ACCUMULATE PART-QTY
END
GET PARTS  << READ FROM DATABASE >>
END.

The following cross-tabulation will be produced:

           PARTS DISTRIBUTION

                PART-LOC
PART-NO     100   101    102     TOTAL PARTS

1785       1000  2000      0            3000

2142        750   100    250            1100

3122          0   100      0             100
```

Dollar-Flow: HP3000 Financial Planning

Jack Damm
The Palo Alto Group
Sunnyvale, CA

The financial planning on the HP3000 with the Dollar-Flow planning language is discussed with particular focus on these three areas: 1) What financial planning is, and why there is a need for computerized planning; 2) Design considerations for friendly user-oriented applications; 3) How the language Dollar-Flow is used for applications such as profit planning.

THE NEED FOR FINANCIAL PLANNING

First, let's start with two questions: What is financial planning? And why is it necessary? Financial planning is making decisions about allocating the scarce resources of an organization so as to best achieve its goals. In the private sector, this usually means how best to allocate money and people to achieve profitability goals. In the public sector, it may mean how best to allocate people and dollars to provide a desired level of service. The main idea here is that the resource is scarce and, as a manager, hard decisions have to be made about how to use it. More specifically, financial planning is setting budgets, making pricing decisions, and estimating future demand for products and services, in order to achieve profit and /or performance goals.

Why is formal planning necessary? First, of course, because a scarce resource (typically money) is involved. If we had enough money for everything, then we could simply raise our salaries and retire early. Secondly, it is very important to have general agreement within an organization about how goals are to be achieved. No assumptions should be made without clearly stating and documenting them. With a good financial plan, trouble signs can be spotted earlier and corrective action taken sooner. Businesses which fail to plan effectively are the best illustration of the need for planning.

Let me offer one last reason why planning is important. For many companies, planning is a necessity because of the complexity of their operations. A typical manufacturing company may purchase thousands of parts for use in a vast variety of products, and assemble them in many different locations. They cannot wait until there is no money in the till to decide that it's time to raise prices. And the current rates of inflation make this an even more important consideration.

THE TYPICAL PLANNING PROCESS

Okay, let's assume that one accepts the need for financial planning. So, what's the big deal? Well let's look at the typical planning process and I'll show you.

First, planning involves lots of numbers. And these numbers change often. Financial planning involves projections into the future and is a very uncertain process. When you're uncertain, then you have to do contingency planning. Play "what if" games. What if sales are 20% higher than planned? What if the cost estimates are too optimistic? What if our product sales mix is different? Because of uncertainty, alternative plans are necessary, increasing the amount of work required to plan several times over.

And that's not all. The attempt to reach a targeted objective such as profit adds to the work. It may take several passes before all of the budgets combined with the sales estimates, cost estimates, and so forth, sum up to the desired results. The task soon becomes monumental.

The following is not an uncommon occurrence: You work many hours preparing budgets and doing sales forecasts. With a board meeting just a few days away, you finish your plan. The company president takes one look at the results of the combined numbers and gives it back, requesting a 15% cut in the budget. You prepare a revised budget, repeat all of the calculations, this time under increasing pressure to get the job done fast. The day before the board meeting, marketing revises the forecast. All of the budgets must be revised again. And now it is getting late into the evening the day before the meeting. Does this seem like a doomsday tale? It's not. I've seen this happen many times. No wonder people dread budgeting time.

Combine the sheer effort required to plan effectively with the requirements for a good plan: It must be TIMELY. In a dynamic, growing company, a plan must reflect today's expectations, not yesterday's. It must be ERROR FREE. Late-night, reworked plans suffer from simple calculation errors. Errors due to using the wrong set of estimates, because they keep on changing. Imagine the embarassment of a summation error. And with all this, the plan must remain FLEXIBLE. I worked on a profit plan for a company a few years ago which added an entire product line between iterations of the plan. And finally, when you are all done, a good plan must be WELL DOCUMENTED. What factors were used for overhead? What was the basis for the final sales figure? How was a particular number calculated? All too often, there is little documentation on how a plan was actually prepared.

To summarize: A typical financial plan involves lots of numbers, which change often. The need for many iterations makes this process time consuming and exhausting. At the same time, the plan must be timely, error free, and well documented. In short, good financial planning is not easy.

WHAT IS THE BEST WAY TO PLAN?

Given that this is the nature of planning, what is the best way to plan? How can it be done with a minimum of difficulty? Traditionally, there have been two ways of planning. Planning by hand (and calculator) and planning using the computer. Let's take a look at both of these methods and evaluate the pluses and minuses of each.

Preparation of plans manually has several drawbacks. First, because of the amount of data involved and the number of iterations, it is slow and time consuming. After many iterations, accuracy becomes a problem. The wrong estimates may be used, particularly if they keep changing. Calculation errors seem to increase with each iteration. And documentation is usually not very good.

On the other hand we have financial planning on the computer using the traditional programming languages like BASIC, FORTRAN, or COBOL. Once set up, a model written in one of these languages will run on the computer in a matter of minutes or seconds. Great! But here's the catch. The model will run very quickly once it has been set up, but it may take months to get it developed. And you need a programmer. Let's see what can happen. You start your plan well in advance of the next budgeting cycle. With six months lead time you give a precise set of specifications to an enthusiastic programmer who dutifully sets about coding your model. At the end of the first three months, he comes back to you with his first try. You patiently point out where the model is not consistent with the specifications, settle on a set of revisions, and the model is reprogrammed to your satisfaction. All set, right? No. As you begin using the model, the company president starts to change his mind (even though he reviewed the original specifications). Add a decimal place here, another line item there. Why aren't all twelve columns of data on the first page? Frustration.

What is the moral of our story? Programming a planning application with the traditional programming languages lacks flexibility. The programmer needs lead time to set up the application and has difficulty in reacting to short term changes. How about adding another division to a multi-divisional company? Try changing every format statement in the model in an hour. And add to that the bother of documentation.

To summarize, manually prepared plans can be flexible, but they take a long time to do and lots of effort, especially if several passes are done. They often lack documentation. Planning with traditional programming languages takes too long to set up, is inflexible, and requires the services of a programmer.

PROBLEM ORIENTED LANGUAGES

Let me digress for a moment. For several decades now, computer scientists have been searching for a "universal" programming language. ALGOL? PL/I? APL? PASCAL? The search goes on. Each has its merits, each its disadvantages. But these "procedure oriented" languages have one thing in common: You have to be a programmer to use them. And it is altogether too easy to include bugs in even the simplest of programs. As long as there is a programmer acting as middleman between the user (or analyst) and the computer there are going to be communication problems. Maintenance problems. Resource and priority problems.

What's the answer? A planning oriented application language which incorporates the good aspects of traditional programming, but eliminates the problems. Where plans can be set up and revised easily, without having to be a programmer. What I am describing here is one example of another class of programming languages, "problem oriented" languages. Languages which have been designed to provide solutions in a general way to classes of problems. Simple enough to be used by non-programmers. Easier to debug. Self-documenting. QUERY is an example of a problem oriented language. It provides access to IMAGE data bases in a fashion simple enough to be used by non-programmers. Dollar-Flow is a problem oriented language, designed as a tool for non-programmers who want to set up tabular planning reports.

Financial planning is an area well suited to problem oriented languages. There is a considerable amount of generality in what planners do, although no two plans are the same. A financial plan typically involves mathematical calculations. And the burden of planning in any other way gives the financial planner considerable incentive to try new approaches.

This is a good start. But we still have to get the planner onto the terminal and communicating with the computer. How is this done? By giving him an effective tool. One which is both friendly and enables him to get the job done in a way that he understands.

DESIGNING FRIENDLY SYSTEMS

This leads us to the next point: What makes a system "friendly"? How can a system be designed so the novice or non-computer type feels comfortable with it? I offer here a few of my ideas and techniques for developing friendly systems.

SIMPLICITY

Keep the system simple at all cost. Do not let the internal structure on the computer dictate how a system looks to the user. Let him express his ideas in his own terms. For example, the original design for the Dollar-Flow language was based on a set of documentation which I prepared for a group of accounting types. This documentation described the workings of a particular customized model on a line by line basis. I figured: What could be a better set of design specifications for a language than actual documentation? As you document your model you are also writing your program! Another example. Dollar-Flow re-orders calculation rules automatically. Thus, line 1 on a report can reference data on line 10, which, in turn, can reference data on line 20. Dollar-Flow automatically figures out the proper sequence for calculations (calculate 20, then 10, then 1) without any intervention by the user.

It is important that the application be self documenting. For example, Dollar-Flow is a menu driven system. At each step of operation, the user knows his alternatives. There is little need for a "pocket guide" to the language. This is not to say that there is no need for manuals. A good manual is important. But it is a fact that few people actually read manuals. The less a system forces a user to read the manual, the more usable it will be.

Not only should the user be told what his alternatives are, the system should also help him to choose the proper response. Throughout the Dollar-Flow prompts, the most likely response is shown in brackets as the "default" response. In some cases, he can use the default response without bothering to even understand the question! For example, the prompt:

```
USE STANDARD OVERALL REPORT FORMAT
(<Y>,N,W-WIDE PAGE)?
```

In one brief prompt, the user can see his options and pick one. A simple carriage return will cause the system to use the default response. And his entire report format is set up. No PRING USING or FORMAT statements. Very simple. And it can be changed easily. As the user becomes more familiar with the language, he can begin to exercise more options. With an 'N' response, Dollar-Flow leads the user through a review of the many formatting alternatives. Report formatting can even be done on a trial and error basis. Start off with the standard format, then change the column width or number of decimal places shown as needs require.

As I already mentioned, the design for the Dollar-Flow calculation rules was based on a set of user oriented documentation. Ask a user to describe how the values on the report are to be calculated in his own terms. With the addition of a few quote marks here and there, he has already written a program in the Dollar-Flow language. Self-documenting languages not only save the effort required for documentation, but make debugging easier as well.

One last comment about simplicity. Save the user concerns about internal structure through structure independent (or data base) approaches to data relationships. One of the beauties of QUERY is that the user doesn't have to concern himself with all of the details of the data base to get a simple report. In Dollar-Flow, all reports are programs, all saved programs are files, and all save files contain reports. To reference data on a saved Dollar-Flow report, simply indicate the line name and the report save file name:

```
MARKETING BUDGET = 'BUDGET' OF 'MKTG'
```

There is no need for the user to know how the data is stored or even which line on the 'MKTG' report is the 'BUDGET' line which he is using.

ERROR HANDLING

Okay, so let's say you have implemented a simple system. Does this mean that users won't make mistakes? Of course not. In fact, the friendlier a system is, the greater the likelihood that the users will not be computer types. So, keep in mind that "To err is human, to forgive is good systems design." Of course, you must edit all inputs. But then use a friendly approach when the user has made an error. Because Dollar-Flow is menu driven, simple typing errors cause the system to repeat the prompt. Errors of a more complex nature, such as where a report is referenced but does not exist, generate intelligible error messages. Along with each error message give a message number. And provide a glossary with the documentation which gives even greater detail on the possible cause of the problem.

At the same time that it is informative, a system should help the user to work around problems. For example, in the case of an invalid report reference in Dollar-Flow, the user can interactively specify a different report name, or values, or zeroes. He can also indicate that computation should cease after a scan for further errors. Again, unless a particular error is extremely serious, warn the user and proceed (with his permission). Another example. As far as the mathematician is concerned, division by zero gives unworkable results. In Dollar-Flow, division by zero yields 'invalid' numbers (which print as asterisks), but doesn't stop computation. It's amazing how much more satisfying a user finds a report filled with asterisks than just a list of error messages. At least he can look at the format to see if it's to his liking.

If you must tell the user that he has made an error, tell him as early as possible. One of the most enlightened things done by the MPE operating system is to edit the job statement when a job is being streamed from an interactive session. It sure is better to find out right away than waiting for the JOB statement. Report development in Dollar-Flow is completely interactive. If a user is setting up a report and he enters a calculation rule with invalid syntax, the system responds with a message immediately, and permits him to edit his error (not unlike the BASIC interpreter). It is not necessary to go into the computation step to find many errors.

MAINTENANCE AND SUPPORT

Let us assume that as an enlightened designer of friendly systems you have now designed and implemented your masterpiece. Are you done? Of course not. This is only the first step. There are two more important aspects which are critical for good, friendly systems: Continuing improvement and good support. Let me talk about continuing development first. No system is great on the first try. I am a believer in the iterative approach to systems development, if you can afford it. I am not talking about sloppy design. I am talking about the tremendous wealth of ideas that you can get from your users, AFTER you have implemented a system. Try to be receptive to the suggestions of your users (even if they are infeasible). Never give a critical user the impression that you think he has just offered a bad idea. Go out of your way to solicit ideas from your users. If the situation merits it, get involved in several of their applications. You can learn about ways the system is being used that you never thought about. Ways in which its use may be awkward. Which messages are more annoying than useful. Which features are badly needed. I send periodic questionnaires to my users (some of them even respond). This helps to prioritize new features. And users group meetings are a great boon to information flow.

How should this wealth of new ideas be integrated into an already developed system? Carefully. Do not rush a new version of a system out to users just because they need a particular feature. You must let a new version of a system be "burned in" first by a test site. Software bugs cost you credibility. Once lost, credibility is very difficult to reestablish, so reliability is extremely important. After all, would a user prefer a system with the bells and whistles he wants but doesn't work, or one which works with a few less features?

Speaking of bugs and user suggestions leads me to the question of support. There is nothing more frustrating to a user than to get 95% of the way to his computer solution only to be stopped by the application package he is using. For any reason. If you can afford to do it, good support pays great dividends. Dollar-Flow is supported in an "on-line" fashion. This means that if a user has a problem, he picks up the telephone and calls. If the problem is with an existing report, we may even log onto his system and take a look at that report. This kind of support not only helps to find and eliminate system problems quickly, but we also find out about areas where the documentation may be confusing (or incorrect). Where another feature might simplify the user's application. In short, on-line support can be another source of good ideas from users.

Let me summarize these techniques for creating friendly systems. First KEEP IT SIMPLE. Try to think like the user instead of a computer expert. Use his terms. Assume that he won't read the manual. Try to make it self-explanatory. Second, be INFORMATIVE but FORGIVING with your error handling. Edit all inputs, but don't bother the user with minor errors. When the application merits, CONTINUING ENHANCEMENT will make a much more usable system. Respond to user suggestions. But exercise good judgment in the trade-off between adding new features and degrading SYSTEM RELIABILITY.

PROFIT PLANNING

I am not going to take too much time on this last part. I am just going to show you a few sample reports which were prepared using Dollar-Flow. (At the risk of violating my agreement not to make a sales pitch, I invite you to visit the PALO ALTO GROUP's booth during the vendor exhibits for a demonstration of Dollar-Flow in action.)

Let me first describe the typical company profit planning cycle and the environment in which a planning tool like Dollar-Flow is used. The typical Dollar-Flow user is the accountant or company controller who is responsible for preparing the reports. Not a programmer. Most users are working on in-house HP3000 systems. With access to CRT's and a system line printer nearby. Reports are written interactively, and manual inputs are also entered via the terminal. Usually, reports are printed on the CRT for review then saved when the user is satisfied with the report. If hard copy is desired, the reports can be routed to the line printer. For generating large numbers of reports, the "batch command mode" is used, where with very little terminal input a large number of reports can be generated.

Profit planning typically begins with a preliminary sales forecast. Preliminary. Sales forecasts always change. And at the last minute, too. Often the sales forecast is done on a product-by-product basis for the first year or so, then combined with overall dollar sales projections further in the future. The near term unit forecasts are sometimes adjusted based on an overall dollar figure. The forecast is iterated several times. To make a change, the product manager just runs Dollar-Flow, inputs whichever figures have changed, pushes a few buttons, and the new sales forecast is ready. Since many parts of the profit plan depend on this sales forecast, the typical plan is usually set up with reports referencing the sales forecast report. If the figures are changed on the sales forecast, these changes will be automatically reflected on the other reports the next time they are run. Some manufacturing companies even use a multi-level sales forecast step, where a build plan (or production plan) is generated from the sales forecast.

Meanwhile, departmental budgets are prepared. Some Dollar-Flow users centralize the budgeting function and only distribute budget worksheets to each department or location. This is usually done if there are only one or two budget iterations. On the other hand, some of our customers distribute the budget preparation, with each location setting up its own budget in Dollar-Flow. In this case, figures can be input to Dollar-Flow, changes can be made, and several iterations of the budget can be done all in a matter of minutes. And budget consolidations are fun! With a few simple commands to Dollar-Flow, a whole series of budgets can be consolidated into a departmental or divisional budget. When changes are made to the low level budgets, they automatically are reflected on the consolidated budget the next time it's run.

The profit/loss projection is next. Using the data from the sales forecast, the build plan, and the budgets, and adding factors for items like sales discounts and returns, a pro forma operating statement is prepared. Often the bottom line (profit) on this report determines what (if any) changes need to be made to the budgets. With a flexible tool like Dollar-Flow, a financial executive can even do sensitivity analysis: What if sales are 20% lower then forecast? What if our discount schedule is more aggressive and our volume is larger?

Some companies that rely on substantial amounts of debt to finance their operations combine the profit/loss projection with a cash flow projection. This is because interest paid (an item of expense on the profit/loss statement) has an impact on the amount of money required to run the business. This determines the level of borrowing, which, in turn, affects the amount of interest which is paid. Dollar-Flow, and most good financial planning languages, can solve the "simultaneous equations" this circular logic represents, and determine a level of debt and debt service which are consistent with each other. This is far more difficult when done manually.

Another procedure which is laborious when done by hand is the aging of accounts receivable and accounts payable projections. Using Dollar-Flow, once the rules for aging have been set up, a change in the sales forecast or the build plan will automatically be reflected in new receipts and payables projections.

And, finally, some companies prepare pro forma balance sheets as the last step in their profit planning cycle. This is not necessarily the way all companies plan. Or even the way all Dollar-Flow users plan. In fact, many Dollar-Flow users are not even responsible for profit planning. Instead, the system is used for a wide variety of ad hoc applications involving calculations on rows and columns of numbers. It is even used as a design tool for systems which will later be hard-coded in COBOL, FORTRAN, or BASIC.

Some of the other applications of Dollar-Flow that I am aware of include:

Product pricing. Comparing alternative prices for a single product (the plotting capability is great for comparisons). Or comparing profit percentage across an entire product line. Financial ratio analysis. Comparing selected financial ratios against industry standards or company objectives. Capital budgeting. Rates of return and discounted cash flows can be calculated easily using built-in financial functions.

Performance reporting. Variance reports showing actual budgets or profits versus plan. How sales are doing against target. (One Dollar-Flow user generates 500 graphs every month showing product line sales performance for every branch of every distributor who markets his products!)

SUMMARY

Let me leave you with a few parting thoughts. Financial planning is not an easy process. Figures change. The whole approach to a plan may change. And you need your results yesterday. Traditional systems design and programming methods are not going to be effective in this kind of situation. Use a better approach. With a friendly, problem oriented planning language like Dollar-Flow, applications nightmares can become applications successes.

FEATURE ARTICLE

FORTRAN/3000 and FORTRAN77: A Comparison

James P. Schwar and Charles L. Best
Lafayette College
Easton, PA 18042

INTRODUCTION

A new standard Fortran was approved as an American National Standard by the American National Standards Institute on April 3, 1978. Since the technical work on this standard was completed in 1977 the designation Fortran 77 is commonly used to distinguish this standard from previous Fortrans. Many, but not all, of the features found in Fortran 77 are incorporated in Fortran/3000. Some of the major new features of Fortran 77 and their relationship to Fortran/3000 are discussed in the following sections.

THE STANDARD

The Fortran 77 standards are found in the publication: American National Standard Programming Language Fortran, ANSI X3.9-1978. This publication presents the detailed specifications for Fortran 77 in 18 sections, with the full language appearing on the righthand pages and subset Fortran appearing on the lefthand pages. It also includes 6 appendices with a summary of Fortran 77 syntax. Subset Fortran will not be considered in the following discussion.

CHARACTER SET

The Fortran 77 character set consists of the twenty-six letters, A through Z, the ten digits 0 through 9, and thirteen special characters.

	blank
=	equals
+	plus
–	minus
*	asterisk
/	slash
(	left parenthesis
)	right parenthesis
,	comma
.	decimal
$	dollar sign
'	apostrophe
:	colon

The quote (") does not appear in the new standard. The collating sequence is A less than B......Z and 0 less than 1......9. There is no collating sequence implied for the special characters. Letters and digits must not be intermixed although all the digits should follow Z or precede A.

STATEMENTS

A line in Fortran 77 is a sequence of 72 characters. Comment lines may contain a C or * in column 1. Continuation lines have not changed and statements are in fixed-field format written only in columns 7 through 72. A statement must contain no more than 1320 characters (Fortran/3000 also allows up to 20 lines for a statement). Statement labels remain as unique positive integers in the range 1 to 99999 in columns 1 through 5. The required order of statements and comment lines is summarized in the Fortran 77 standards by the following chart:

Comment Lines	FUNCTION, SUBROUTINE or BLOCK DATA Statement		
	FORMAT and ENTRY statements	PARAMETER statements	IMPLICIT statements
			other specification statements
		DATA statements	statement functions
			executable statements
END statement			

Vertical lines delineate statements that may be interspersed, while horizontal lines delineate statements that must not be interspersed. The END statement, as in Fortran/3000 is now executable. This chart essentially reflects the statement order in Fortran/3000.

Arithmetic statements are handled essentially as in Fortran/3000. Variable names are still limited to six alphabetic or numberic characters, the first of which must be alphabetic. Data types are: integer, real, double precision, complex, logical and character.

The character data type corresponds to the CHARACTER specification in Fortran/3000. There is, however, a difference in handling substrings. Fortran 77 substring designators specify the leftmost and rightmost character respectively in the substring. For example A(2:4) specifies characters two through four of character variable A, and B(4,3)(1:6) specifies characters one through six of the character array element B(4,3). Note that square brackets are not used. The concatenation operator for the character variable is the double slash // in Fortran 77. Partial-word designators are not part of Fortran 77.

Arrays in Fortran 77 may have up to seven dimensions and allow both an upper and lower bound to be specified in the dimensioning statement. If the lower bound is omitted, the default value is one.

DIMENSION A(-1:8),B(10,10),C(8)

The one-dimensional A array has elements from -1 to 8, where the fourth element is A(2), while the fourth element of the one-dimensional C array is C(4). B is a two-dimensional array whose first element begins at 1,1.

Subscripts must be integer expressions and a subscript may itself be a subscripted variable. Fortran 77 does not permit real subscripts as is the case in Fortran/3000.

Control statements form the major difference between Fortran 77 and Fortran/3000. Fortran 77 permits the block IF statement which is an implementation of the IFTHENELSE programming structure familiar to SPL, BASIC and COBOL users. The general form of this statement is

```
IF (logical expression) THEN
          block
ELSE
          block
ENDIF
```

Optionally, the ELSE block may be omitted.

The DO statement (or DO loop) has been modified to allow integer, real, or double precision data types not only for the indexing parameters, as currently available in Fortran/3000, but also for the DO variable. For example

```
DO 10 A = 1.0,0.0,-0.1
```

is now valid in Fortran 77. Execution of a DO loop proceeds essentially as in Fortran/3000, however, the execution follows the DOWHILE programming structure with initialization outside the loop and incrementation (stepping) at the end of the loop.

List directed input/output is part of Fortran 77. An allowable format for list directed input/output permits replacing the format statement number with an asterisk. In general, input/output statements have been expanded to include statements such as OPEN, CLOSE and INQUIRE, as well as the more familiar READ, WRITE, BACKSPACE, ENDFILE and REWIND found in Fortran/3000. The OPEN statement, an auxiliary input/output statement, connects an existing file to a unit (similar to FOPEN intrinsic). The CLOSE statement, another auxiliary input/output statement, terminates the connection (similar to FCLOSE). The auxiliary statement INQUIRE permits a determination about the properties of a particular file (similar to FGETINFO). The reader is referred to section 12.10 of the Fortran 77 Standards for details on the auxiliary input/output statements. Several minor additions have been made to the format specifications, most notable of which are the edit descriptors:

(1) TL (tab left) and TR (tab right);

(2) S, SP and SS used to control option plus characters in numeric field output;

(3) BN and BZ used to specify interpretation of blanks other than leading blanks;

(4) the A or Aw is used for character data, where w is the field width.

The Ow, Zw, Nw.d, Mw.d, Rw and S edit descriptors familiar to the 3000 Fortran programmers are not part of Fortran 77.

FUNCTIONS and SUBROUTINES

The ENTRY statement and RETURN n are now supported in Fortran 77. A SAVE statement, which allows local variables, local arrays and labelled common blocks to be saved between calls to subroutines and functions has been added in Fortran 77. Section 15.10 of the Fortran 77 Standards lists these intrinsics. Most of the generic names are the same as those found in Fortran/3000. An exception is that Fortran 77 includes the intrinsic for arcsine and arccosine. There are a few minor differences between Fortran/3000 and Fortran 77 in the naming of generic functions.

CLOSURE

The full impact of Fortran 77 can only be appreciated by a careful examination of the new standards. There are many improvements, both major and minor, to the language. Fortran/3000 contains many of the features associated with Fortran 77. The major areas of difference are the:

(1) ability to specify a lower bound for an array;
(2) block IF statement;
(3) extended DO statement.

REFERENCES

1. HP3000 Computer System, Fortran Reference Manual 30000-90040.
2. American National Standard Programming Language FORTRAN X3.9-1978.

LIBRARY CORNER

LIBRARY ALERT

If you are an HP3000 Series I user [Installation Member] or your site has not yet installed the 1906 IT on your Series II or III, you may have a problem with Release 06 of the Contributed Library. The Library is produced using 4K blocking factor, while your machine is expecting 1K.

If you are a Series I user or do not have an SPL compiler, the tape sent to you would have been in the right format except that we do not know who you are! Your solution is to call the Executive Office at (301) 768-4187 and a usable tape will be sent to you at no charge in exchange for your present Release 06 tape.

The HPGSUG Board policy requires that each release of the Contributed Library will be compatible with the most current HP operating system installed at the majority of members sites. Therefore, for Series II and III users who have not installed the 1906 IT, a replacement tape of the Release 06 may be purchased for the nominal fee of $25. Notify the Executive Office, return the originial Release 06 and a compatible tape will be forwarded to you.

If you are an HP3000 user [Installation member] and do not have an SPL compiler, you will have a problem with Release 06 of the Contributed Library. Based on information given HPGSUG by HP at the time Release 06 was being prepared, the decision was made to use SPL for the Library utilities as SPL was the only compiler distributed with HP's Fundamental Operating System [FOS]. As SPL is no longer included with HP's FOS, those sites which do not have an SPL compiler will be provided a PROG version of the Contributed Library [at no Charge] by notifying the Executive Office and returning your orginial Release 06.

For future releases of the Contributed Library, due to the above change and other changes by HP, which include Run Time Library, the Library Committee will reconsider the type of code placed on the Contributed Library for Release 07.
Please note: Allow two weeks for delivery for any replacement tape.

Software Exchange at San Jose

Wayne Holt

An organized software exchange will take place at the February, 1980 North American meeting of the User Group in San Jose, California. This is your chance to share tips, techniques, and that special program you've been telling everyone about. Participation is simple as long as you observe the following rules:

1. Contributors should try to follow the standard HPGSUG Library guidelines when submitting their software. This includes:
   * standard naming conventions for files and groups, i.e., SOURCE, PUB, DOC, DATA, and JOB;
   * a signed software contribution release form;
   * a copy of extended documentation if necessary.

   Check the INFOBASE account on the library for instructions.

2. All submissions MUST be in :STORE format. The name of the account is SANJOSE and the creator of all files MUST be in MGR.

3. Text files are welcome. Although they will not be placed in the HPGSUG Library, they will be of interest to those sites that participate in the swap.

4. Any individual contributing to the Exchange will receive a copy of the Swap tape. Depending upon the size of the swap, every effort will be made to return your tape *before* the end of the convention. *This is subject to available labor and computer time!*

5. An attempt will be made to submit the software in the Exchange to the HPGSUG Library. This will be successful only if the contributions follow submission standards and are well documented. *Include your name and address in your DOC file!*

6. All contributions are due as soon as possible after registration. Contributions made after Monday will be handled on an as-time-allows basis. All tapes submitted should be clearly marked as to your name, address, and hotel room while at the Convention. It is strongly advised that you use a 2400 ft. reel of tape, since some swaps have been "hefty" in size.

7. If you do not wish to contribute, but still want a copy of the tape, an effort will be made to accomodate you on a serve yourself, first in - first out basis. In the closing hours of the Convention, *if time allows,* a special job will be set up to make copies. There are no guarantees!

8. PLEASE. Remember that this is a volunteer project for the convenience of the Group members. Time and available labor are scarce; the only promise that we can make is that we shall try our best. With your help and participation we can make the software exchange at San Jose the best ever!

See you in San Jose February 25th thru 29th

TIPS & TECHNIQUES

Hints on IOSTAT2

Robert M. Green

The SCRUG79 swap tape contains a very interesting program called IOSTAT2, but no documentation for the program. After trying it and talking to a user who had used an earlier version of the program, I was able to deduce the following operating instructions. This program appears to be perfectly safe, but it does use privileged mode to access the I/O tables.

IOSTAT2: REAL-TIME I/O MONITOR

VERSION: 1.06
SOURCE: Not Available
INSTALL: Copy to an account/group with capabilities like PUB.SYS. UTIL. SYS is suggested.

HISTORY: This program was found on the SCRUG79 swap tape. There was no documentation; what follows was discovered via phone calls to an anonymous source. The program has only been run on the 1906 MIT by us.

PURPOSE: IOSTAT2 allows you to monitor the I/O activity of your HP3000 (MPEIII) by taking "snapshots" of the I/O Queue Table. You can see which logical device numbers have the most activity, and how much disc activity the memory manager is causing (MAM). You can see if all disc drives are setting equal activity (if not, perhaps you should move some files). You can examine selected functions (READ, WRITE, FOPEN, FCLOSE) and selected programs (via the PIN).

EXAMPLE (your replies are enclosed in (( ))).

```
    :run iostat2
    IOSTAT2(TEW) VERSION 01.06 05 JAN79

    (C) HEWLETT-PACKARD COMPANY 1979

    SPECIAL HEADINGS (Y OR N)?
      y
    ENTER: LDEV,HDG -OR- DRIVER,HDG
    ) 14,Iss
    ) 1, sys
    )
```

((As you will see below, IOSTAT2 classifies each logical devise into a class. To request a different classification, you enter the commands above. I have requested that device 14 be separated as "ISS" disc, and device 1 be separated as "SYS" disc. You could separate each disc to see how disc accesses are distributed.
))

```
    FREE OR IN USE TO BE EXAMINED (F OR I): f
```

((The IOQ table is used to "queue" input/output requests for all devices. The size of this table is specified in SYSDUMP. At any given moment in time, some of the entries will be "IN USE" for I/O requests that are waiting or in progress. The rest will be FREE (released from previous requests). To examine queuing problems, answer "I" above; to examine overall activity, answer "F" above.
))

```
    PIN TO BE EXAMINED: return
```

((Carriage return selects all processes. Or, you can enter a selected PIN (process id number) to monitor. See the SHOWQ command, or the OVERLORD program. This option is not used in this example.
))

```
    SPECIFIC FUNCTION? return
```

((Carriage return means all functions. I think the possible functions are READ, WRITE, FOPEN, FCLOSE. If you select a specific function, you can see what percentage of the activity it reads, versus writes. Since FOPENs are very expensive, you can also use this to monitor opens.
))

```
HDG CNT       LDEVS

UNUSED  13 15 16 17 18 19
DISC  6 2 3 4 5 11 12
 LP   1 6
 MT   4 7 8 9 10
TERM 48 20 21 22 23 24 25
        26 27 28 29 30....
 ISS 1 14
 SYS 1 1
```

((IOSTAT2 prints your heading table, showing which logical devices are included in each heading class.
))

((IOSTAT2 now begins sampling the IOQ at 15-20 second intervals (the interval can be changed; see below). After each sample; one line of summary is printed.
))

```
TIME        COUNT     FREE     EXMND
09:06:41               40        40
09:06:55     334       40        40
09:07:20     636       39        39
09:07:48     710       43        43
09:08:11     547       43        43
09:08:33     578       41        41
```

```
MAM     ISS     SYS     DISC     TERM
 26      4       29      2        5
 12      8       10     15        7
 24      4       20     11        4
 31      7       23      4        9
 32      1       31      4        7
 22      6       22      2       11
```

((Column headings have this meaning:

TIME: time of the sample

COUNT: total I/O requests since last sample. If this number is greater than the FREE columns, you are only examining a portion of the I/O requests.

FREE: total free entries in the table (or INUSE, total inuse entries in the table).

EXMND: how many of the free entries were included in this summary line; some entries may be left from a previous sample period when the system is not busy

MAM: how many of the requests (of all specific functions, I think) were on behalf of Memory Management this includes reading in code segments from the System SL (LDEV1) and other SLs, and program files, and swapping data segments (LDEV1).

ISS: how many requests on the ISS disc (LDEV14); could include some MAM requests.

SYS: how many requests on the LDEV1; mostly MAM requests and directory searches.

DISC: how many requests on the other 79xx disc drives; could include MAM requests.

TERM: other device type.

Please note that in the first sample, 26 of the 35 disc requests were for MAM. This machine needs memory.
))

```
***CONTROL Y*** control y
```

((I hit control-y on the terminal.))

```
CONTINUE? n

END OF PROGRAM
```

((I stopped the program by answering "N". But, there are other options.
))

Robert M. Green
ROBELLE CONSULTING LTD.
#130-5421 10th Avenue
Delta, B.C. V4M 3T9
Canada (604) 943-8021

SPECIAL REPORTS

Bug/Enhancement Poll

Ross Scroggs

The Interface Committee of HPGSUG has initiated the polling procedures through which the members of the Users Group can indicate which defects in current HP software products critically affect the operation of their HP3000. Included in each issue of the **JOURNAL** will be a prestamped return card on which you will list your most critical bugs. The responses will be collected and forwarded to HP for their consideration. HP plans on using this information as one of the variables in the bug prioritization function. The results of each poll will be published in a subsequent issue of the **JOURNAL.**

Included on the poll card is space for an enhancement request. You should include a brief description of an enhancement you would like to see in a current HP software, hardware, service, or other product. These responses will be classified and distributed to the appropriate HP personnel. These requests, along with those received through other means, should provide HP with information concerning the direction they might want to take when planning future enhancements. These requests may also be used to formulate questionnaires that will address specific product areas.

HP will respond in subseqüent issues of the **JOURNAL** as to how they are using the information gathered in the polls and will provide general indications as to what direction they are taking with regards to specific problems or products.

The Interface Committee urges you to complete the enclosed poll card as soon as possible so that the results can be provided to HP in a timely manner.

Please note that this poll does not replace the normal bug reporting mechanism, you should promptly report all new bugs through the regular bug reporting procedure. The purpose of this poll is to determine the number of people affected by a particular bug.

Editor's Note: The BUG/Enhancement Poll Card is found on the back cover of the **JOURNAL.** Clip and mail today! (No postage is required if mailed in the United States.)

HP General Systems Users Group MEETING UPDATE

DONN PARKER TO GIVE KEYNOTE ADDRESS

Donn B. Parker will deliver the keynote address at the Hewlett-Packard General Systems Users Group 1980 North American Meeting. Parker, an internationally renowned expert in the field of computer crime, is a senior management systems consultant at SRI International, Menlo Park, California. He specializes in the area of computer abuse research, security in computer systems, programming methodology, and management of computer services. He is the author of the report Computer Abuse, prepared for the National Science Foundation and has written what is considered the definitive book on computer security, *Crime By Computer*.

Make your plans now to attend the 1980 HPGSUG North American Meeting and hear this outstanding specialist in our industry.

KEEP IT RUNNING

Two years ago the Bay Area Regional Users Group sponsored a user-oriented Hardware Familiarization and Maintenance Course with the HP Technical Support Group. Those of us involved in the presentation are excited for we have implemented the skills learned from the orginial course which our theme says most succinctly: 'Keep It Running'.

All of us in the Data Processing industry must depend on computers for our "daily bread." Thus, the intention of these seminars is to pass along information to the HP3000 user which will enable him/her to deal effectively with, or even more importantly, avoid completely computer problems.

The series participants will be shown the most effective way to deal with problems by utilizing user-run diagnostics. There will be an opportunity to find out the most intimate details of hardware and software architecture, and the intricacies of system dispatching and queuing. Learn everything you ever wanted to know about saving data, file system security, and the "How To's" of placing effective service calls. See what makes "Mr. Goodbyte" (your CE) tick, and meet the technical experts who back him up.

The 1980 series, presented by HP-Neely Santa Clara's Technical Support Group, under the direction of Frank Steiner and Ed Canfield, will be relocated to San Jose to man the 'experts' tables and give you a glimpse of what goes on 'behind the HP service scenes'.

You will find the presentations the answer to your questions about "keeping it running," and will provide the keys to making yours a smoother operation.

HP SYSTEMS ON-SITE

e Systems Managers of the 1980 North American ers Group Meeting are Jack Craig of Medi-Data Services in Santa Cruz, and Dave Moirao of Longs Drug Stores in Walnut Creek. Jack and Dave, and their system operators, will preside over the Hewlett-Packard computer systems from HP's General Systems Division, providing computational power to vendors who need system access to display and demonstrate their products. Additionally, access will be available to Meeting attendees who want "hands-on" time on the systems.

The HP250 systems installed will feature 160 to 448 Kbytes of random access memory and up to 39.2 Megabytes of disc storage. This system also utilizes a double density flexible disc drive with 1.2 Megabytes of storage.

Also, the HP300 will be on display. It occupies a small physical space but provides powerful computing capabilities. The system features up to 1 Megabytes of disc storage. Like the HP250, the HP300 utilizes a double density flexible disc drive for system utility, system backup, and offline storage of data.

An HP3000 Series 30 is the newest addition to the HP3000 family. This is HP's entry level system which features up to 1024 Kybtes of random access memory and up to 960 Megabytes of disc storage. The Series 30 can support up to 32 on-line terminals, from 1 to 4 magnetic tape drives, a 100 to 400 lpm (line per minute) printer, an integrated flexible disc drive with 1.2 Megabytes of storage, and remote diagnostic capability.

The HP3000 Series IIIs, each featuring an expandable memory configuration from 256 Kbytes to 2048 Kbytes of random access memory and from 50 to 960 Megabytes of disc storage. Each system can support up to 64 on-line terminals, 8 magnetic tape drives, and up to 4 printers with speeds from 300 to 1000 lpm.

Peripheral systems on display include: 2621A, 2621P, 2635A, 2645A, 2647A, 2648A, 2649E and 3076A terminals; the 2608A, 2619A and 2631A printers; the 7970B and 7970E magnetic tape drives; the 7906A and 7925A disc drives; and the 7310A and 9872B plotters.

Housing Information

The cost of housing is **NOT** included in the cost of the Meeting Registration fee. A separate form is included for housing. Duplicate it as necessary where additional rooms are required by several attendees. Please fill it out completely and submit it to the San Jose Convention Bureau whose address appears on the form. Do not include your housing request form or any payment for housing with your Meeting pre-registration.

The hotels listed here have agreed to make a certain number of rooms available at a special convention rate; these will be provided on a first-come, first-served basis. However, in order to make it possible for the Housing Bureau to meet hotel deadlines, your housing request form must be **received no later than Juanuary 30, 1980.** After this date, any available rooms will be released by the hotels to general occupancy at their higher standard rates, and the Housing Bureau will no longer be involved in handling reservations for our meeting. As a consequence, housing reservations after January 30 are to be made directly by attendees as best they can. Telephone numbers are provided in the event you miss the January 30 deadline.

HOTEL	SINGLE RATE	DOUBLE RATE
Holiday Inn Park Center Plaza (Adjacent to the Convention Center) (408) 998-0400	$33 (King Room $37)	$37 (King Room $41)
Holiday Inn Airport(San Jose) (408) 287-5340	$32	$36
Hyatt House Hotel (Meeting Headquarters) (408) 298-0300	$46	$50
Le Baron Hotel (408) 288-9200	$42	$48
Vagabond Motel (San Jose) (408) 294-8138	$36	$42

Room Reservation Form

DEADLINE: JANUARY 30, 1980

Please Reserve: ____ SINGLE ____ DOUBLE ____ DOUBLE/DOUBLE (Phone reservations not accepted)

Hotel/Motel Preference: 1st Choice __________ 2nd __________ 3rd __________

Dates of: Arrival __________; ________ *After 6 P.M.?* Departure __________

A DEPOSIT OF $45^{00}(U.S.) FOR EACH ROOM MUST ACCOMPANY RESERVATION. ONE RESERVATION PER REQUEST FORM.

Make check(s) payable and send to: **San Jose Convention Bureau/HPGSUG**
P.O. Box 6178, San Jose, CA 95150 U.S.A.

NAME ____________________ **PHONE** ____________________

ADDRESS ____

STATE(COUNTRY) ____________________ **POSTAL CODE** __________

COMPANY ____________________

Registration Fees

$200. U.S. when this form is **postmarked** no later than December 31, 1979. Registrations postmarked after that date will be assessed a $40.00 balance-due fee at the door. Please make check payable to **HP GSUG 1980 NORTH AMERICAN MEETING.** Payments from locations outside of the USAshould be drawn in US funds on your bank's correspondent U.S. bank.

Registration Form

Please print — leaving one space between words.

Last Name Initial First Name

Title

Company

Street Address

City State/Province

Country Zip or Postal Code

(Duplicate this form for additional persons registering for the meeting.)

Send Registration form to:

HEWLETT-PACKARD GENERAL SYSTEMS USERS GROUP
1980 North American Meeting
P.O. Box 3010
Stanford, California U.S.A. 94305
February 25-29, 1980

HP GENERAL SYSTEMS USERS GROUP

INFORMATION THROUGH INTERFACE AND INVOLVEMENT

APPLICATION FOR MEMBERSHIP

(Copy this form for additional names)

(Please print or type)

NAME ______________________________
(Voting individual)

TITLE ______________________________

COMPANY ______________________________

STREET ADDRESS ______________________________

CITY ______________ STATE/PROVINCE ______________ ZIP ________

COUNTRY ______________________________ POSTAL CODE ________

TELEPHONE NO. (   ) ______________________

TYPE OF MEMBERSHIP (check items desired)

[ ] General Member — Total
$20 Annual Dues . . . . $ 20
Includes:
1. Journal
2. Newsletter

[ ] Copies of Master Contributed Library and one Update.
__________ x $200 . . . . $______
no. of copies
Check media type:
[ ] 800 bpi or [ ] 1600 bpi

[ ] Additional subscription(s) to the Journal and Newsletter to the same name and address
__________ x $15 . . . . $______
no. of copies
Total $______

[ ] Installation Member — Total
$200 New Member Dues . . . . $ 200
(Annual renewal $150)
Includes:
1. Journal
2. Newsletter
3. Contributed Library
4. Voting Privileges
5. Conference Proceedings

Check media type:
[ ] 800 bpi or [ ] 1600 bpi

[ ] Additional subscription(s) to the Journal and Newsletter to the same name and address
__________ x $15. . . . . $______
Total $______

If you are a member of a Regional Users Group, please specify ______________________

TYPE OF MACHINE (check one)

[ ] HP 3000 [ ] HP 300 [ ] HP 250

Are you interested in having your name on:

1) A commercial mailing list? . . . . [ ] yes [ ] no

2) A membership roster circulated to other members for non-commercial purposes? [ ] yes [ ] no

[ ] CHECK ENCLOSED [ ] PURCHASE ORDER # ______________ ENCLOSED

Payment should be made to "HP General Systems Users Group". Payments from organizations outside of the United States must be drawn in U.S. Funds and on your bank's correspondent U.S. bank.

As a condition of membership, I hereby agree not to distribute the Contributed Library to any person without the prior written approval by the HP General Systems Users Group.

__________ Date ______________________ Signature

EMPIRE TOWERS 7300 RITCHIE HWY. GLEN BURNIE, MARYLAND 21061 301-768-4187

HP General Systems Users Group
Membership Price Schedule
Effective January 1, 1980
(One Year Basis)

General Member
New: $20
Renewal: $20
Includes:
1. Journal Subscription*
2. Newsletter Subscription

Installation Member
New: $200
Renewal: $150
Includes:
1. Journal Subscription*
2. Newsletter Subscription
3. Contributed Library
4. Voting Privileges

* Note: Both General and Installation Members are entitled to additional subscriptions of the Journal and Newsletter to the same address for $15/subscription.

Any individual or organization may purchase the Contributed Library for $200.

------------------------------ FOLD HERE ------------------------------

Return Address

____
____
____
____

HP General Systems Users Group
Empire Towers
7300 Ritchie Highway
Glen Burnie, MD 21061
U.S.A.

ATTN: Rella M. Hines
Executive Director

------------------------------ FOLD HERE ------------------------------

Clip along dotted line

BUG/ENHANCEMENT POLL

NAME: ______________________________

PHONE: (   ) ______________________

TELEX: ______________________________

COMPANY: ______________________________

ADDRESS: ______________________________

____

____

Please indicate the BUGS that critically affect the operation of your HP3000. Include any number of bugs but indicate only those with greatest impact. The bugs should be identified by their Known Problem Report (Service Request) number as found in the most recent Software Status Bulletin. Please include BUGS with an "open" status only.

____

____

____

____

____

____

____

Please include one brief ENHANCEMENT Request (hardware/software/service/other) you would like HP to address. Be brief and include a Keyword that classifies your request as specifically as possible, e.g., 7925 Disc/COBOL/CE support.

KEYWORD: ______________________________

REQUEST: ______________________________

____

____

____

____

____

____

____

HP General Systems Users Group
Empire Towers
7300 Ritchie Highway
Glen Burnie, Maryland
21061, USA

RELLA M. HINES, EXECUTIVE DIRECTOR
(301) 768-4187

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 83
GLEN BURNIE, MD.

LINFORD HACKMAN
VYDEC, INC.
9 VREELAND RD.
FLORHAM PARK, NEW JERSEY
07932, USA

Clip along dotted line

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS **PERMIT NO. 8** **GLEN BURNIE, MD.**
POSTAGE WILL BE PAID BY ADDRESSEE

HP General Systems Users Group
Empire Towers
7300 Ritchie Hwy.
Glen Burnie, MD. 21061