INFORMATION CROSSROADS OF THE 80s

SEPTEMBER 8 - 13, 1985 INTEREX

HOSTED BY BALTIMORE/WASHINGTON RUG

PROCEEDINGS

HP 3000/SERIES 100

VOLUME II

V2

INTEREX

the International Association of
Hewlett-Packard Computer Users

Proceedings

of the

1985 CONFERENCE

at

Washington, D.C.

Hosted by the
Baltimore-Washington Regional
Users Group

Papers for the
HP3000
and
Series 100

VOLUME II
PAPER 3043-3091

Sam Inks, Editor

[illegible]

VOLUME II

Introduction

This volume of the Proceedings of the INTEREX 1985 North American Conference was printed from machine readable text supplied by the authors (with a few exceptions). Each paper was formatted in TDP and printed on an HP2680A Laser Printer.

Thanks go to the authors who sent their papers in on time and in the requested formats. Special thanks to the Review Committee for all of their time, efforts and suggestions.

A special thanks also to those who have helped me keep my sanity, typed the non-machine readable papers, held meetings at their house or in some other manner lent their own time and support to the publishing of these proceedings.

REVIEW COMMITTEE	SPECIAL PEOPLE
Nick Demos	Dean Gabersek
Sam Inks	Millie Gabersek
Suzanne Perez	John Grether
Joan Peters	Dorothy Inks
Kevin Rhea	Lee Mauck
Chris Seiger	Mary Moorer
	Nancy Murray
	Ron Smirlock

I would also like to offer special thanks to Jim Cummins, my boss, and to Atlantic Research Corporation for allowing me the time to participate in this undertaking and for the use of the HP Computer systems.

My thanks also to F. Stephen Gauss of the HP1000 group for his help and support.

Index by Author

Vol.

Index by Title

3043. SIMPLE STEPS TO OPTIMIZE TRANSACT/3000 APPLICATIONS

TOM NIELSEN
MWC ATC
HP-NAPERVILLE

****

The trend towards higher labor costs and lower hardware costs in today's economy has led to a rapid increase in the use of a class of computer productivity tools called "Fourth Generation Languages (4GL's)." 4GL's are designed to relieve the programmer of much of the tedious coding required by many traditional languages, allowing the programmer to code at a much higher, and productive level. This reduction in programmer effort results in the inherent tradeoff that the computer is now faced with the task of doing more of the work in order to make the application run. This increased burden on the computer resources invariably causes the topic of performance ramifications to be brought up in nearly every discussion of 4GL's. While 4GL's inherently use more computer resources than conventional languages, there are many techniques the programmer can use to attempt to minimize this difference.

The following paper attempts to outline numerous techniques that are avaliable to programmers creating TRANSACT/3000 (Hewlett Packard's version of a 4GL) applications which will improve their program's efficiency significantly with minimal effort. The format will be to first discuss the generalities of each performance optimizing technique in light of TRANSACT/3000's methodology. This general discussion will then be substantiated with short test programs designed to highlight the performance ramifications of various techniques.

Since many of the test programs utilize an IMAGE data base, a familiarity with that data base may be helpful in the analysis of certain portions of code. For this reason, a schema listing of the IMAGE data base TESTB has been included as APPENDIX B. Listings of the test programs and their results are also included in APPENDIX B. APPENDIX A includes an explanation of the testing procedures used.

****

I. MINIMIZE DATA ACCESSED FROM IMAGE DATA BASE FILES

This statement goes beyond the obvious of advising against unnecessary accesses to a data file and stresses the optimization of each data access verb (ie. GET, FIND, OUTPUT, PUT). Since TRANSACT data access verbs actually call IMAGE intrinsics directly, this point can be used to improve performance for programs coded in any language.

When a program accesses a data item, the IMAGE intrinsic must first determine that item's data base relative item number from the data base root file. The security matrix (also in the IMAGE root file) must then be checked to verify that the password used to open the data base grants the appropriate access to that item. These two overhead operations must be performed for each item requested by the program for each access. Understandably, then, the fewer items the program attempts to access, the less time it will take for the IMAGE intrinsics to transfer the data to/from the program's DATA register.

Unfortunately, this relatively simple step is often bypassed by many programmers. Reasons seem to vary from ignorance of the functions of the LIST= option to the philosophy that it's just easier to retrieve all the data so that whatever the program may need down the road will be available in the DATA register. The former could be resolved with adequate training and experience, and the latter could be resolved with good initial program design.

The TRANSACT Reference Manual makes it very clear that the only items absolutely required in the LIST= option are those items that are being used as MATCH criteria in selecting specific records. Perhaps the biggest misconception is that key or search items must be included in the LIST= option of retrieval verbs. This is entirely false using TRANSACT, and differs from the rules present when calling the IMAGE intrinsics directly from ordinary languages. In fact, for most keyed access retrievals, the program already has the desired value (we had to use it to set up the Key and Argument registers). Who knows how many millions of unnecessary item retrievals are being done on search items alone.

Programs T1 and T2 highlight the performance implications of retrieving unnecessary data from an IMAGE data set. Both programs center around a FIND(SERIAL) command which retrieves 150 detail records. The only difference between the two is that T1 retrieves values for all six items in the data set, while T2 retrieves values for only one item. An initial GET is inserted to remove the irrelevant overhead of opening the dataset from the timed loop. The PERFORM= paragraph performs a data base access to eliminate any optimization of the list parameter that either TRANSACT or IMAGE may attempt on the iterative FIND verb.

```
*******************************************************************

PROGRAM              # OF ITEMS           RELEVANT
 NAME                RETRIEVED            ELAPSED
                                           TIME
************         **********           ********
T1                       6                  2999
T2                       1                  2516

*******************************************************************
```

II. STRUCTURE THE LIST= OPTION TO OPTIMIZE PERFORMANCE

Performance optimization suggestion I dealt with the idea of minimizing the number of items which are accessed from an IMAGE data base. Once the programmer has decided on the items which must be accessed, however, there are further simple steps which can be taken to improve performance. These steps deal with the structuring of the LIST= option of data access verbs.

As the TRANSACT Reference Manual indicates, there are numerous methods to indicate which items are to be accessed. The most common of these seem to be:

1.	LIST=(item1:itemn)	requests retrieval of values for all items in the LIST register between item1 and itemn, inclusive.
2.	LIST=(item1,..,itemn)	requests retrieval of values for the specific items listed.
3.	LIST=(@)	a feature as of A.02.02 which specifies all items in the data set.

While the second option is often simpler to implement, as it does not require any strategic management of the LIST register, it is also less efficient as far as program execution. Specifying the LIST= option as an item range (as in example #1), on the other hand, will result in faster execution, but is more difficult to manage the LIST register, as the programmer must attempt to put items that are accessed at the same time contiguously in the LIST register. The idea of different data access verbs requesting a different combination of items leads to complication of this process. Performance tests indicate that specifying LIST=(@) is identical to specifying all items via an item range, assuming that range includes all items.

Many programmers agonize over how to build the program's LIST register so that they may always use the more efficient item range construct. It should be noted that minimization of data access should take precedence over this concern and thus the LIST register should be constructed to allow the item range construct to be used in the most frequent data access verbs (use test modes or temporary DISPLAY statements to determine the most frequent verbs). The remainder of the data access verbs, then, should simply use an item list construct. To attempt to manipulate the LIST register to allow item ranges for all verbs would be quite laborious, probably end up slowing execution, and would cause many debugging nightmares.

Programs T9, T3, and T4 highlight the performance ramifications of different LIST= constructs. Program T9 uses an item range, T3 uses an item list, and T4 uses a variation on the item range that is even a little faster than the traditional item1:itemn construct. T4 uses the itemn construct which specifies retrieval values for all items in the LIST register from the bottom of the LIST register through itemn. This technique turns out to be a little more efficient because of reduced search time of the linked list (see performance suggestion III). It should be noted here that while the performance difference may be small in the test programs supplied, these programs only reference six (6) items. The performance ramifications would obviously become more substantial as the number of items accessed increased.

****

PROGRAM NAME	LIST= CONSTRUCT	RELEVANT ELAPSED TIME
********	*********	*******
T9	(ITEM1:ITEMN)	2665
T3	(ITEM1,ITEM2,...,ITEMN)	2863
T4	(:ITEMN)	2659

****

III. PLACE FREQUENTLY USED ITEMS NEAR THE TOP OF THE LIST REGISTER

Every time a program references a data item's value, TRANSACT must first locate the corresponding item name in the LIST register, which then maps to the appropriate location in the DATA register. Since TRANSACT allows multiple occurrences of the same item name in the LIST register, TRANSACT can not use a direct, or hashed, search method to locate the item in the LIST register. Instead, TRANSACT maintains the items in the LIST register as a linked list which is searched whenever a data item name is referenced. The search will continue down the linked list until one of two conditions occur:

1. The item name being searched for is found, providing a map into the DATA register.
2. The linked list is exhausted without finding the item name being searched for, in which case the run-time error "ITEM NOT FOUND IN LIST REGISTER" will occur.

Obviously, we are more interested in case #1, as case #2 is merely a programming error which must be corrected.

Understanding the direction of the search is crucial to optimizing a program's performance. If the programmer can minimize the time taken by each search (which, remember, occurs every time a data item is referenced) the program will, as a direct result, take less time to execute. As it turns out, the search starts out at the top of the LIST register (ie. it starts with the item most recently put into the LIST register) and works its way to the beginning (or bottom) of the LIST register. In order to minimize search times, then, frequently accessed items should be added to the LIST register last. This optimization technique is even more crucial for applications using a large number of data items, as search times can become significant.

Programs T5 and T6 point out the performance implications of putting frequently used items near the top of the LIST register. Both programs have a relatively large LIST register (129 items). Program T5 accesses an item near the top of the LIST register 200 times while T6 accesses an item at the bottom of the LIST register. Note the significant difference in execution time between the two otherwise identical programs.

```
*****************************************************************

PROGRAM            POSITION OF          NUMBER OF       RELEVANT
 NAME              ITEM IN LIST         ACCESSES        ELAPSED
                                                         TIME
*******           *************         *********       *******
T5                TOP (#128)             200             1490
T6                BOTTOM (#1)            200             1827

*****************************************************************
```

IV. REDUCE TRANSACT'S MANAGEMENT OF THE WORK AREA

This performance optimizing suggestion is perhaps the least frequently thought of by TRANSACT programmers. With each TRANSACT application comes an area called the WORK AREA, which is a temporary work area containing the MATCH, UPDATE, and INPUT registers. Every time the program utilizes these special registers, an entry is put into this work area, which is implemented as a linked list. Whenever an entry is deleted from these registers (ie. via RESET(OPTION)), TRANSACT merely flags the entry in the linked list as deleted, but does not return that space to the reusable state. With repeated use of these registers, then, the WORK AREA, which has a user specified size (via the WORK=option on the SYSTEM statement), will soon become full of "deleted" entries. When this occurs, TRANSACT will automatically call a routine called REWORK, which will actually delete the previously flagged entries and return that space to the available list.

In order to improve program performance, the programmer should attempt to minimize the number of times REWORK must be called. This can be achieved by increasing the size of the WORK AREA via the WORK= option of the SYSTEM statement. Increasing the WORK AREA, however, will increase stack usage at run time, so it must be done with care. Perhaps a realistic approach is to attempt to minimize the calling of REWORK , while at the same time monitoring stack usage to keep it appropriate for your specific system.

Test modes 102 and 123 should be used in making these adjustments to the WORK AREA. Test mode 102 reports statistics on WORK AREA usage during execution, while test mode 123 prints a warning message to TRANOUT everytime REWORK is called.

Programs T7 and T8 highlight the difference in program execution time when we minimize reorganization of the WORK AREA by increasing its size. T7 has a small work area and requires 124 reorganizations of the WORK AREA, while T8 only requires 6, hence exectution time is decreased.

****

PROGRAM NAME	WORK AREA SIZE	REWORK CALLED	RELEVANT ELAPSED TIME
*******	*********	*******	*********
T7	15 WORDS	124	2181
T8	255 WORDS	6	2151

****

V. OPTIMIZE NUMERIC CALCULATIONS

TRANSACT was intended as a transaction processing tool as well as a prototyping tool. Unfortunately, TRANSACT is not terribly efficient when it comes to things like number crunching, because of some data type conversion operations performed. A few simple techniques, however, can at least make any necessary numeric calculations a little less time consuming.

Perhaps the most common suggestion for streamlining TRANSACT number crunching is to have the TRANSACT application invoke (via the PROC verb) some external routine written in a more efficient language to do the calculations for TRANSACT. This suggestion, however, does have a couple of weaknesses. It first assumes that another programming language is available on the system, and second it assumes that all number crunching is done at one time in the program. It does not take into account the idea of "sporadic" calculations done throughout the TRANSACT application. The suggestions presented below will help optimize TRANSACT applications in both of these instances.

The first suggestion is to choose an appropriate data type for elements that will be involved in numeric calculations. The chart below indicates Real (R)and Integer (I) are the most efficient data types for numeric calculations, while Zone Decimal type (Z) is the worst. If your application permits then, attempt to make all variables that will be used extensively in calculations type R or I. Counter items are an excellent example of items which should be streamlined.

The second suggestion to minimize the amount of overhead incurred by TRANSACT numeric calculations is to make all elements involved the same type and decimal scale if possible. This will reduce the amount of type conversion that TRANSACT must do before the calculations (done by a procedure called CALCULATE). An example is programs T10, T15, and T16. T10 crunches two I(5,0,2) elements in 1446 relative CPU seconds; T15 crunches two R(6,0,4) fields in 1446 relative CPU seconds. But when we crunch one item of I(5,0,2) and one of R(6,0,4) together as we do in T16, note the increase to 2332, which is higher than the calculation using two of the less efficient Z type variables.

The chart below, then, depicts nine programs which do 100 relevant numeric calculations with the only difference being the type and decimal scale of variables used.

****

PROGRAM NAME	VARIABLE #1	VARIABLE #2	MAXIMUM OPERAND VALUE	RELEVANT ELAPSED TIME
*****	********	********	********	********
T10	I(5,0,2)	SAME AS #1	32,767	1446
T11	I(10,0,4)	SAME AS #1	2,147,483,648	1446
T12	Z(4,0,4)	SAME AS #1	9,999	2171
T13	Z(8,0,8)	SAME AS #1	99,999,999	2187
T14	P(7,0,4)	SAME AS #1	9,999,999	1491
T15	R(6,0,4)	SAME AS #1	999,999	1446
T16	I(5,0,2)	R(6,0,4)		2332
T17	I(10,0,4)	I(7,2,4)		2455
T18	P(7,0,4)	P(7,2,4)		1511

****

VI. OPTIMIZE INTERNAL DATA TRANSFERS

Transact offers two verbs to be used in transferring values from one data item to another. Unfortunately, many people seem to be unclear, or misinformed, about the difference between these two verbs and in which situations to use either. The following discussion should clear up these misconceptions of TRANSACT's MOVE and LET verbs, and points out how the appropriate use of the MOVE verb can improve performance.

TRANSACT's MOVE verb is specifically designed for straight data transfer from one location in the DATA register to another and for alphanumeric operations. Unfortunately. the TRANSACT reference manual's discussion almost leads you to believe that the MOVE verb can only be used when manipulating (transferring or concatenating) alphanumeric values. To many people's surprise, however, the MOVE verb can also be used to transfer numeric values. There are, however, certain limitations inherent in the MOVE verb:

1. No numeric calculations can be done with the MOVE verb.

2. The MOVE verb does not do data type conversions, so both source and destination variables should be the same type.

Because of the fact that MOVE does no data type conversions, it is a relatively fast verb for data transfer. Programmers should thus use the MOVE verb for both alphanumeric and numeric data transfer operations between like type data elements. Only for those operations involving incompatible data elements and/or numeric calculations should the LET verb be used.

Programs T20 and T21 point out a very simple example of the increased performance of using MOVE over LET. T20 performs 100 MOVE operations, moving an I(9) value to another I(9) field. T21 uses the LET verb instead of the MOVE.

****

PROGRAM NAME	VERB USED	RELEVANT ELAPSED TIME
********	****	********
T20	MOVE	759
T21	LET	800

r**

VII. USE FILE VERB FOR MPE FILE OPERATIONS

Transact offers two methods for reading, writing, updating, and sorting MPE files. Unfortunately, because of the fact that most programmers are more familiar with the data management verbs (ie. PUT, GET etc) used with IMAGE files, they use these less efficient verbs when interacting with MPE files as well. Because of the fact that these verbs were designed to work with the more complex data base files, as well as to be generic enough to work with all types of files, they incur much overhead which is not necessary in working with MPE files.

The more efficient method for interacting with MPE files is the FILE verb. This verb has modifiers which allow programmers to read, write, update, or sort MPE files. As the sample program indicates below, the FILE verb is much more efficient than the data management verbs. Unfortunately, the FILE verb does not utilize TRANSACT`s special registers, so its uses are limited to operations such as a straight serial read or simple additions to the file.

****

PROGRAM NAME	VERB USED	FREQUENCY	RELEVANT ELAPSED TIME
********	******	*********	*********
T22	GET(SERIAL)	200	4385
T23	FILE(READ)	200	3451
T24	PUT	200	5144
T25	FILE(WRITE)	200	4277

****

SUMMARY

The seven techniques outlined aboved, then, are presented in order to help programmers who are developing TRANSACT applications to better understand TRANSACT's methodology in certain operations. The intent was to present these ideas in a way that would allow programmers to weigh alternatives available to them in order to maximize their application's performance. It should be noted that the list is by no means exhaustive in presenting methods of improving performance. Additional considerations are outlined in Appendix E of the TRANSACT Reference Manual. In addition to the techniques presented here, as well as those outlined in Appendix E, the programmer should keep in mind the ordinary techniques in program design, in attempting to design an optimal application.

APPENDIX A

The test programs presented in this paper were all created by the author. The attempt was made to make the programs as simple as possible while still allowing the point to be made, and may thus often seem very trivial at first glance. That was the intention. Attempts were made programmatically to insure a "clean" compare between multiple test programs (ie. beginning many programs with an initial disc access to insure initial disc head placement etc.). Most of these techniques are discussed directly in the paper where appropriate.

The methodology selected in determining performance implications was one of at least two methods which could have been used. The reader will note that most programs initially place an item labelled TIMESTART into the LIST register, initializing it to PROCTIME. The code which follows, then, is the code that is being compared between programs, followed immediately by the placement of a second item, labelled TIMESTOP, into the LIST register, again initializing it to PROCTIME. The difference between TIMESTART and TIMESTOP, then, is the amount of CPU milliseconds within the loop, and is reported as ELAPTIME. This ELAPTIME value, then, can be used for comparison purposes.

A second method of determining performance ramifications would be to use TRANSACT's test mode 4 to determine actual instruction timings. This method seems to result in the same performance guidelines being agreed upon, while the testing procedure is much more laborious, hence the first method was selected for use with this paper.

The test programs were run on a dedicated HP3000 Series 48 with disc caching enabled.

APPENDIX B

DATA BASE SCHEMA FOR TESTB

```
BEGIN DATA BASE TESTB;

PASSWORDS:

ITEMS:
     E1,                X2        ;
     E2,                X2        ;
     E3,                X2        ;
     E4,                Z4        ;
     E5,                Z4        ;
     E6,                Z8        ;
     E7,                I1        ;
     K1,                X2        ;

SETS:

NAME:   MASTERSET,      MANUAL    ;
ENTRY:  K1             (1);
CAPACITY: 50;

NAME:   DETAILSET,      DETAIL    ;
ENTRY:  K1             ( MASTERSET       ),
        E1,
        E2,
        E3,
        E4,
        E5,
        E6,
        E7;
CAPACITY: 150;

END.
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T1

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000              SYSTEM T1,BASE=TESTB(";",1);
     2.000  0000        << LOOP THROUGH DETAIL WITH ALL ITEMS RETRIEVED >>
     3.000  0000        DISPLAY "ALL ITEMS BEING RETRIEVED";
     7.000  0002        LIST K1:E2:E3:E4:E5:E6:ELAPTIME;
     9.000  0012
    10.000  0012           LIST TIMESTART,PROCTIME;
    11.000  0015           FIND(SERIAL) DETAILSET,LIST=(K1,E2,E3,E4,E5,E6),
                               PERFORM=A;
    12.000  0026           LIST TIMESTOP,PROCTIME;
    13.000  0029
    14.000  0029           LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    15.000  0031           DISPLAY ELAPTIME;
    16.000  0033        END;
    17.000  0034
    18.000  0034        A:
    19.000  0034        GET(SERIAL) DETAILSET,LIST=(E2,E3);
    20.000  0040        RETURN;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:03
   ELAPSED TIME=00:00:04

*******************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ALL ITEMS BEING RETRIEVED

ELAPSED TIME
 3030

EXIT/RESTART(E/R)?>
ALL ITEMS BEING RETRIEVED

ELAPSED TIME
 2999

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T2

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000           SYSTEM T2,BASE=TESTB(";",1);
     2.000  0000     << LOOP THROUGH DETAIL RETRIEVING ONE ITEM >>
     3.000  0000     DISPLAY "ONE ITEM RETRIEVED";
     7.000  0002     LIST K1:E2:E3:E4:E5:E6:ELAPTIME;
     9.000  0012
    10.000  0012        LIST TIMESTART,PROCTIME;
    11.000  0015        FIND(SERIAL) DETAILSET,LIST=(E2),PERFORM=A;
    12.000  0019        LIST TIMESTOP,PROCTIME;
    13.000  0022
    14.000  0022        LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    15.000  0024        DISPLAY (ELAPTIME);
    16.000  0026     END;
    17.000  0027
    18.000  0027     A:
    19.000  0027     GET(SERIAL) DETAILSET,LIST=(E2,E3);
    20.000  0033     RETURN;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:03
   ELAPSED TIME=00:00:03

**********************************************************************

 TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ONE ITEM RETRIEVED

ELAPSED TIME
 2570

EXIT/RESTART(E/R)?>
ONE ITEM RETRIEVED

ELAPSED TIME
 2516

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T3

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T3,BASE=TESTB(";",1);
    3.000  0000    DISPLAY "ITEM LIST BEING RETRIEVED";
    7.000  0002    LIST K1:E2:E3:E4:E5:E6:ELAPTIME;
    9.000  0012
   10.000  0012       LIST TIMESTART,PROCTIME;
   11.000  0015       FIND(SERIAL) DETAILSET, LIST=(K1,E2,E3,E4,E5,E6),
   11.500                PERFORM=A;
   12.000  0026       LIST TIMESTOP,PROCTIME;
   13.000  0029
   14.000  0029       LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   15.000  0031       DISPLAY ELAPTIME;
   16.000  0033    END;
   17.000  0034    A:
   18.000  0034       GET(SERIAL) DETAILSET,LIST=(E2);
   19.000  0037       RETURN;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:04

**********************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ITEM LIST BEING RETRIEVED

ELAPSED TIME
 2894

EXIT/RESTART(E/R)?>
ITEM LIST BEING RETRIEVED

ELAPSED TIME
 2863

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T4

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T4,BASE=TESTB(";",1);
    3.000  0000    DISPLAY "IMPLIED ITEM RANGE SPECIFIED";
    7.000  0002    LIST K1:E2:E3:E4:E5:E6:ELAPTIME;
    9.000  0012
   10.000  0012       LIST TIMESTART,PROCTIME;
   11.000  0015       FIND(SERIAL) DETAILSET,LIST=(:E6),PERFORM=A;
   12.000  0019       LIST TIMESTOP,PROCTIME;
   13.000  0022
   14.000  0022       LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   15.000  0024       DISPLAY ELAPTIME;
   16.000  0026    END;
   17.000  0027    A:
   18.000  0027       GET(SERIAL) DETAILSET,LIST=(E2);
   19.000  0030       RETURN;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:07

********************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
IMPLIED ITEM RANGE SPECIFIED

ELAPSED TIME
 2691

EXIT/RESTART(E/R)?>
IMPLIED ITEM RANGE SPECIFIED

ELAPSED TIME
 2659

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T5

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

                 SYSTEM T5,DATA=2048,300;
         0000    DISPLAY "PROG TO ACCESS ITEM ON TOP OF LIST REG";
         0002    DEFINE(ITEM) COUNT I(4);
         0002    LIST E2:
         0003          ELAPTIME:
         0004          E1:
         0005          E1:
         0006          E1:
         0007          E1:
         0008          E1:
         0009          E1:
         0010          E1:
         0011          E1:
         0012          E1:
         0013          E1:
         0014          E1:
         0015          E1:
         0016          E1:
         0017          E1:
         0018          E1:
         0019          E1:
         0020          E1:
         0021          E1:
         0022          E1:
         0023          E1:
         0024          E1:
         0025          E1:
         0026          E1:
         0027          E1:
         0028          E1:
         0029          E1:
         0030          E1:
         0031          E1:
         0032          E1:
         0033          E1:
         0034          E1:
         0035          E1:
         0036          E1:
         0037          E1:
         0038          E1:
         0039          E1:
         0040          E1:
         0041          E1:
         0042          E1:
         0043          E1:
         0044          E1:
```

```
47.000  0045          E1:
48.000  0046          E1:
49.000  0047          E1:
50.000  0048          E1:
51.000  0049          E1:
52.000  0050          E1:


53.000  0051          E1:
54.000  0052          E1:
55.000  0053          E1:
56.000  0054          E1:
57.000  0055          E1:
58.000  0056          E1:
59.000  0057          E1:
60.000  0058          E1:
61.000  0059          E1:
62.000  0060          E1:
63.000  0061          E1:
64.000  0062          E1:
65.000  0063          E1:
66.000  0064          E1:
67.000  0065          E1:
68.000  0066          E1:
69.000  0067          E1:
70.000  0068          E1:
71.000  0069          E1:
72.000  0070          E1:
73.000  0071          E1:
74.000  0072          E1:
75.000  0073          E1:
76.000  0074          E1:
77.000  0075          E1:
78.000  0076          E1:
79.000  0077          E1:
80.000  0078          E1:
81.000  0079          E1:
82.000  0080          E1:
83.000  0081          E1:
84.000  0082          E1:
85.000  0083          E1:
86.000  0084          E1:
87.000  0085          E1:
88.000  0086          E1:
89.000  0087          E1:
90.000  0088          E1:
91.000  0089          E1:
92.000  0090          E1:
93.000  0091          E1:
94.000  0092          E1:
95.000  0093          E1:
96.000  0094          E1:
97.000  0095          E1:
   98.000  0096          E1:
   99.000  0097          E1:
  100.000  0098          E1:
  101.000  0099          E1:
  102.000  0100          E1:
  103.000  0101          E1:
  104.000  0102          E1:
  105.000  0103          E1:
  106.000  0104          E1:
  107.000  0105          E1:



  108.000  0106          E1:
  109.000  0107          E1:
  110.000  0108          E1:
  111.000  0109          E1:
  112.000  0110          E1:
  113.000  0111          E1:
  114.000  0112          E1:
  115.000  0113          E1:
  116.000  0114          E1:
  117.000  0115          E1:
  118.000  0116          E1:
  119.000  0117          E1:
  120.000  0118          E1:
  121.000  0119          E1:
  122.000  0120          E1:
  123.000  0121          E1:
  124.000  0122          E1:
  125.000  0123          E1:
  126.000  0124          E1:
  127.000  0125          E1:
  128.000  0126          E1:
  129.000  0127          E1:
  130.000  0128          E1:
  131.000  0129          E1;
  132.000  0130
  133.000  0130      LIST COUNT,INIT:TIMESTART,PROCTIME;
  134.000  0135
  135.000  0135      REPEAT
  136.000  0135 1    MOVE (E1)="AB";  << ACCESS ITEM AT TOP OF LIST >>
  137.000  0137 1    LET (COUNT)=(COUNT)+1;
  138.000  0140 1    UNTIL (COUNT)=200;
  139.000  0143      LIST TIMESTOP,PROCTIME;
  140.000  0146      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
  141.000  0148      DISPLAY ELAPTIME;
  142.000  0150      END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:07
  ELAPSED TIME=00:00:09
```

```
**********************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
PROG TO ACCESS ITEM ON TOP OF LIST REG

ELAPSED TIME
 1490

EXIT/RESTART(E/R)?>
PROG TO ACCESS ITEM ON TOP OF LIST REG

ELAPSED TIME
 1490

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T6

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T6,DATA=2048,300;
    2.000  0000    DISPLAY "PROG TO ACCESS ITEM ON BOTTOM OF LIST REG";
    3.000  0002    DEFINE(ITEM) COUNT I(4);
    4.000  0002    LIST E2:
    5.000  0003          ELAPTIME;
    6.000  0004          E1:
    7.000  0005          E1:
    8.000  0006          E1:
    9.000  0007          E1:
   10.000  0008          E1:
   11.000  0009          E1:
   12.000  0010          E1:
   13.000  0011          E1:
   14.000  0012          E1:
   15.000  0013          E1:
   16.000  0014          E1:
   17.000  0015          E1:
   18.000  0016          E1:
   19.000  0017          E1:
   20.000  0018          E1:
   21.000  0019          E1:
   22.000  0020          E1:
   23.000  0021          E1:
   24.000  0022          E1:
   25.000  0023          E1:
   26.000  0024          E1:
   27.000  0025          E1:
   28.000  0026          E1:
   29.000  0027          E1:
   30.000  0028          E1:
   31.000  0029          E1:
   32.000  0030          E1:
   33.000  0031          E1:
   34.000  0032          E1:
   35.000  0033          E1:
   36.000  0034          E1:
   37.000  0035          E1:
   38.000  0036          E1:
   39.000  0037          E1:
   40.000  0038          E1:
   41.000  0039          E1:
   42.000  0040          E1:
   43.000  0041          E1:
   44.000  0042          E1:
   45.000  0043          E1:
```

```
46.000  0044          E1:
47.000  0045          E1:
48.000  0046          E1:
49.000  0047          E1:
50.000  0048          E1:
51.000  0049          E1:
52.000  0050          E1:


53.000  0051          E1:
54.000  0052          E1:
55.000  0053          E1:
56.000  0054          E1:
57.000  0055          E1:
58.000  0056          E1:
59.000  0057          E1:
60.000  0058          E1:
61.000  0059          E1:
62.000  0060          E1:
63.000  0061          E1:
64.000  0062          E1:
65.000  0063          E1:
66.000  0064          E1:
67.000  0065          E1:
68.000  0066          E1:
69.000  0067          E1:
70.000  0068          E1:
71.000  0069          E1:
72.000  0070          E1:
73.000  0071          E1:
74.000  0072          E1:
75.000  0073          E1:
76.000  0074          E1:
77.000  0075          E1:
78.000  0076          E1:
79.000  0077          E1:
80.000  0078          E1:
81.000  0079          E1:
82.000  0080          E1:
83.000  0081          E1:
84.000  0082          E1:
85.000  0083          E1:
86.000  0084          E1:
87.000  0085          E1:
88.000  0086          E1:
89.000  0087          E1:
90.000  0088          E1:
91.000  0089          E1:
92.000  0090          E1:
93.000  0091          E1:
94.000  0092          E1:
95.000  0093          E1:
96.000  0094          E1:
  97.000  0095         E1:
  98.000  0096         E1:
  99.000  0097         E1:
 100.000  0098         E1:
 101.000  0099         E1:
 102.000  0100         E1:
 103.000  0101         E1:
 104.000  0102         E1:
 105.000  0103         E1:
 106.000  0104         E1:
 107.000  0105         E1:


 108.000  0106         E1:
 109.000  0107         E1:
 110.000  0108         E1:
 111.000  0109         E1:
 112.000  0110         E1:
 113.000  0111         E1:
 114.000  0112         E1:
 115.000  0113         E1:
 116.000  0114         E1:
 117.000  0115         E1:
 118.000  0116         E1:
 119.000  0117         E1:
 120.000  0118         E1:
 121.000  0119         E1:
 122.000  0120         E1:
 123.000  0121         E1:
 124.000  0122         E1:
 125.000  0123         E1:
 126.000  0124         E1:
 127.000  0125         E1:
 128.000  0126         E1:
 129.000  0127         E1:
 130.000  0128         E1:
 131.000  0129         E1;
 132.000  0130
 133.000  0130     LIST COUNT,INIT:TIMESTART,PROCTIME;
 134.000  0135
 135.000  0135     REPEAT
 136.000  00135 1  MOVE (E2)="AB";  <<ACCESS ITEM AT BOTTOM OF LIST >>
 137.000  0137 1   LET (COUNT)=(COUNT)+1;
 138.000  0140 1   UNTIL (COUNT)=200;
 139.000  0143     LIST TIMESTOP,PROCTIME;
 140.000  0146     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
 141.000  0148     DISPLAY ELAPTIME;
 142.000  0150     END;

CODE FILE STATUS: REPLACED
```

```
0 COMPILATION ERRORS
PROCESSOR TIME=00:00:07
  ELAPSED TIME=00:00:08

**********************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
PROG TO ACCESS ITEM ON BOTTOM OF LIST REG

ELAPSED TIME
 1856

EXIT/RESTART(E/R)?>
PROG TO ACCESS ITEM ON BOTTOM OF LIST REG

ELAPSED TIME
 1827

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T7

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000            SYSTEM T7,BASE=TESTB(";",1),WORK=15;
    2.000  0000      DEFINE(ITEM) TIMESTART I(9):
    3.000  0000                   TIMESTOP  I(9):
    3.100  0000                   COUNT  I(4):
    4.000  0000                   ELAPTIME  I(9);
    5.000  0000      LIST K1:E2:E3:E4:E5:E6:ELAPTIME:COUNT;
    6.000  0008      MOVE (E2)="AA";
    7.000  0010      LET (COUNT)=1;
    7.100  0012      LIST TIMESTART,PROCTIME;
    8.000  0015      WHILE (COUNT) <250
    9.000  0015      DO
   10.000  0015 1       SET(MATCH) LIST(E2);
   10.100  0020 1       RESET(OPTION) MATCH;
   11.000  0021 1       LET (COUNT)=(COUNT)+1;
   12.000  0024 1    DOEND;
   13.000  0026      LIST TIMESTOP,PROCTIME;
   14.000  0029      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   15.000  0031      DISPLAY ELAPTIME;
   16.000  0033      END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:04

*****************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>

ELAPTIME:
 2185

EXIT/RESTART(E/R)?>

ELAPTIME:
 2181

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T8

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T8,BASE=TESTB(";",1),WORK=255;
    2.000  0000    DEFINE(ITEM) TIMESTART I(9):
    3.000  0000                 TIMESTOP  I(9):
    3.100  0000                  COUNT I(4):
    4.000  0000                 ELAPTIME  I(9);
    5.000  0000    LIST K1:E2:E3:E4:E5:E6:ELAPTIME:COUNT;
    6.000  0008    MOVE (E2)="AA";
    7.000  0010    LET (COUNT)=1;
    7.100  0012    LIST TIMESTART,PROCTIME;
    8.000  0015    WHILE (COUNT) <250
    9.000  0015    DO
   10.000  0015 1     SET(MATCH) LIST(E2);
   10.100  0020 1     RESET(OPTION) MATCH;
   11.000  0021 1     LET (COUNT)=(COUNT)+1;
   12.000  0024 1  DOEND;
   13.000  0026    LIST TIMESTOP,PROCTIME;
   14.000  0029    LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   15.000  0031    DISPLAY ELAPTIME;
   16.000  0033    END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:05

*******************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>

ELAPTIME:
 2156

EXIT/RESTART(E/R)?>

ELAPTIME:
 2151

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T9

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000         SYSTEM T9,BASE=TESTB(";",1);
    3.000  0000   DISPLAY "ALL ITEMS BEING RETRIEVED";
    7.000  0002   LIST K1:E2:E3:E4:E5:E6:ELAPTIME;
    9.000  0012
   10.000  0012      LIST TIMESTART,PROCTIME;
   11.000  0015      FIND(SERIAL) DETAILSET,LIST=(K1:E6),PERFORM=A;
   12.000  0020      LIST TIMESTOP,PROCTIME;
   13.000  0023
   14.000  0023      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   15.000  0025      DISPLAY ELAPTIME;
   16.000  0027   END;
   17.000  0028   A:
   18.000  0028      GET(SERIAL) DETAILSET,LIST=(E2);
   19.000  0031      RETURN;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:03

 *******************************************************************

TRANSACT/3000    HP32247A.01.07 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ALL ITEMS BEING RETRIEVED

ELAPSED TIME
 2693

EXIT/RESTART(E/R)?>
ALL ITEMS BEING RETRIEVED

ELAPSED TIME
 2665

EXIT/RESTART(E/R)?>
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T10

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T10;
    2.000  0000    DISPLAY "INTEGER ARITHMETIC";
    3.000  0002    DEFINE(ITEM) COUNT I(9):I1 I(5,0,2):I2 I(5,0,2);
    3.100  0002    LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008    LIST TIMESTART,PROCTIME;
    6.000  0011    REPEAT
    7.000  0011    LET (I1)=(I1)+(I2);
    8.000  0013 1  LET (COUNT)=(COUNT)+1;
    9.000  0016 1  UNTIL (COUNT)=100;
   10.000  0019    LIST TIMESTOP,PROCTIME;
   11.000  0022    LIST ELAPTIME;
   12.000  0023    LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025    DISPLAY ELAPTIME;
   14.000  0027    SET(COMMAND) INITIALIZE;
   15.000  0029    END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:03
```

****

```
TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
INTEGER ARITHMETIC

ELAPSED TIME
1446
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T11

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

                SYSTEM T11;
       0000     DISPLAY "DOUBLE WORD INTEGER ARITHMETIC";
       0002     DEFINE(ITEM) COUNT I(9):I1 I(10,0,4):I2 I(10,0,4);
       0002     LIST COUNT,INIT:I1,INIT:I2,INIT;
       0008     LIST TIMESTART,PROCTIME;
       0011     REPEAT
       0011     LET (I1)=(I1)+(I2);
       0013 1   LET (COUNT)=(COUNT)+1;
       0016 1   UNTIL (COUNT)=100;
       0019     LIST TIMESTOP,PROCTIME;
       0022     LIST ELAPTIME;
       0023     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
       0025     DISPLAY ELAPTIME;
       0027     SET(COMMAND) INITIALIZE;
       0029     END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:03

************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
DOUBLE WORD INTEGER ARITHMETIC

ELAPSED TIME
 1446
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T12

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T12;
    2.000  0000    DISPLAY "ZONED ARITHMETIC";
    3.000  0002    DEFINE(ITEM) COUNT I(9):I1 Z(4,0,4):I2 Z(4,0,4);
    3.100  0002    LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008    LIST TIMESTART,PROCTIME;
    6.000  0011    REPEAT
    7.000  0011    LET (I1)=(I1)+(I2);
    8.000  0013 1  LET (COUNT)=(COUNT)+1;
    9.000  0016 1  UNTIL (COUNT)=100;
   10.000  0019    LIST TIMESTOP,PROCTIME;
   11.000  0022    LIST ELAPTIME;
   12.000  0023    LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025    DISPLAY ELAPTIME;
   14.000  0027    SET(COMMAND) INITIALIZE;
   15.000  0029    END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:03

************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ZONED ARITHMETIC

ELAPSED TIME
 2171
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T13

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000           SYSTEM T13;
    2.000  0000     DISPLAY "ZONED ARITHMETIC";
    3.000  0002     DEFINE(ITEM) COUNT I(9):I1 Z(8,0,8):I1 Z(8,0,8);
    3.100  0002     LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008     LIST TIMESTART,PROCTIME;
    6.000  0011     REPEAT
    7.000  0011     LET (I1)=(I1)+(I2);
    8.000  0013 1   LET (COUNT)=(COUNT)+1;
    9.000  0016 1   UNTIL (COUNT)=100;
   10.000  0019     LIST TIMESTOP,PROCTIME;
   11.000  0022     LIST ELAPTIME;
   12.000  0023     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025     DISPLAY ELAPTIME;
   14.000  0027     SET(COMMAND) INITIALIZE;
   15.000  0029     END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:03

**********************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
ZONED ARITHMETIC

ELAPSED TIME
2187
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T14

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000            SYSTEM T14;
    2.000  0000      DISPLAY "PACKED ARITHMETIC";
    3.000  0002      DEFINE(ITEM) COUNT I(9):I1 P(7,0,4):I2 P(7,0,4);
    3.100  0002      LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008      LIST TIMESTART,PROCTIME;
    6.000  0011      REPEAT
    7.000  0011      LET (I1)=(I1)+(I2);
    8.000  0013 1    LET (COUNT)=(COUNT)+1;
    9.000  0016 1    UNTIL (COUNT)=100;
   10.000  0019      LIST TIMESTOP,PROCTIME;
   11.000  0022      LIST ELAPTIME;
   12.000  0023      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025      DISPLAY ELAPTIME;
   14.000  0027      SET(COMMAND) INITIALIZE;
   15.000  0029      END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:03

**********************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
PACKED ARITHMETIC

ELAPSED TIME
1492
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T15

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000            SYSTEM T15;
    2.000  0000      DISPLAY "REAL ARITHMETIC";
    3.000  0002      DEFINE(ITEM) COUNT I(9):I1 (6,0,4):I2 R(6,0,4);
    3.100  0002      LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008      LIST TIMESTART,PROCTIME;
    6.000  0011      REPEAT
    7.000  0011      LET (I1)=(I1)+(I2);
    8.000  0013 1    LET (COUNT)=(COUNT)+1;
    9.000  0016 1    UNTIL (COUNT)=100;
   10.000  0019      LIST TIMESTOP,PROCTIME;
   11.000  0022      LIST ELAPTIME;
   12.000  0023      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025      DISPLAY ELAPTIME;
   14.000  0027      SET(COMMAND) INITIALIZE;
   15.000  0029      END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:03

*************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
REAL ARITHMETIC

ELAPSED TIME
1446
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T16

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T16;
    2.000  0000    DISPLAY "INTEGER AND REAL ARITHMETIC";
    3.000  0002    DEFINE(ITEM) COUNT I(9):I1 (5,0,2):I2 R(6,0,4);
    3.100  0002    LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008    LIST TIMESTART,PROCTIME;
    6.000  0011    REPEAT
    7.000  0011    LET (I1)=(I1)+(I2);
    8.000  0013 1  LET (COUNT)=(COUNT)+1;
    9.000  0016 1  UNTIL (COUNT)=100;
   10.000  0019    LIST TIMESTOP,PROCTIME;
   11.000  0022    LIST ELAPTIME;
   12.000  0023    LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025    DISPLAY ELAPTIME;
   14.000  0027    SET(COMMAND) INITIALIZE;
   15.000  0029    END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:04

************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
INTEGER AND REAL ARITHMETIC

ELAPSED TIME
2332
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T17

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000         SYSTEM T17;
    2.000  0000   DISPLAY "DIFFERENT SCALE INTEGER DECIMAL
                              ARITHMETIC";
    3.000  0002   DEFINE(ITEM) COUNT I(9):I1 I(5,0,2):I2 I(7,2,4);
    3.100  0002   LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008   LIST TIMESTART,PROCTIME;
    6.000  0011   REPEAT
    7.000  0011   LET (I1)=(I1)+(I2);
    8.000  0013 1 LET (COUNT)=(COUNT)+1;
    9.000  0016 1 UNTIL (COUNT)=100;
   10.000  0019   LIST TIMESTOP,PROCTIME;
   11.000  0022   LIST ELAPTIME;
   12.000  0023   LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025   DISPLAY ELAPTIME;
   14.000  0027   SET(COMMAND) INITIALIZE;
   15.000  0029   END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:05

*************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
DIFFERENT SCALE INTEGER DECIMAL ARITHMETIC

ELAPSED TIME
2454
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T18

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000           SYSTEM T18;
    2.000  0000     DISPLAY "DIFFERENT SCALE PACKED ARITHMETIC";
    3.000  0002     DEFINE(ITEM) COUNT I(9):I1 P(7,0,4):I2 P(7,2,4);
    3.100  0002     LIST COUNT,INIT:I1,INIT:I2,INIT;
    5.000  0008     LIST TIMESTART,PROCTIME;
    6.000  0011     REPEAT
    7.000  0011     LET (I1)=(I1)+(I2);
    8.000  0013 1   LET (COUNT)=(COUNT)+1;
    9.000  0016 1   UNTIL (COUNT)=100;
   10.000  0019     LIST TIMESTOP,PROCTIME;
   11.000  0022     LIST ELAPTIME;
   12.000  0023     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
   13.000  0025     DISPLAY ELAPTIME;
   14.000  0027     SET(COMMAND) INITIALIZE;
   15.000  0029     END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:04
  ELAPSED TIME=00:00:03

*************************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>
DIFFERENT SCALE PACKED ARITHMETIC

ELAPSED TIME
1511
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T20

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000           SYSTEM T20;
    1.100  0000     << PROGRAM TO COMPARE MOVE AND LET STATEMENTS >>
    2.000  0000     DEFINE(ITEM) I1 I(9):I2 I(9);
    3.000  0000     LIST I1,INIT:I2,INIT:ELAPTIME,INIT:COUNT,INIT;
    4.000  0008     LIST TIMESTART,PROCTIME;
    4.100  0011     REPEAT
    5.000  0011     MOVE (I2)=(I1);
    5.010  0013 1   LET (COUNT)=(COUNT)+1;
    5.020  0016 1   UNTIL (COUNT)=100;
    5.100  0019     LIST TIMESTOP,PROCTIME;
    5.200  0022     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    5.300  0024     DISPLAY ELAPTIME;
    7.000  0026     END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:04

***********************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>

ELAPSED TIME
 759
```

COMPILE LISTING AND RUN RESULTS OF PROGRAM T21

```
COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

    1.000          SYSTEM T21;
    1.100  0000    << PROGRAM TO COMPARE MOVE AND LET STATEMENTS >>
    2.000  0000    DEFINE(ITEM) I1 I(9):I2 I(9);
    3.000  0000    LIST I1,INIT:I2,INIT:ELAPTIME,INIT:COUNT,INIT;
    4.000  0008    LIST TIMESTART,PROCTIME;
    4.100  0011    REPEAT
    5.000  0011    LET (I2)=(I1);
    5.010  0013 1  LET (COUNT)=(COUNT)+1;
    5.020  0016 1  UNTIL (COUNT)=100;
    5.100  0019    LIST TIMESTOP,PROCTIME;
    5.200  0022    LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    5.300  0024    DISPLAY ELAPTIME;
    7.000  0026    END;

CODE FILE STATUS: REPLACED


0 COMPILATION ERRORS
PROCESSOR TIME=00:00:03
  ELAPSED TIME=00:00:03

*********************************************************************

TRANSACT/3000    HP32247A.01.09 - (C) Hewlett-Packard Co. 1983

SYSTEM NAME>

ELAPSED TIME
 800

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF T22

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000              system T22,file=MPEFILE(READ(OLD));
     2.000  0000        << PROGRAM TO TRAVERSE THROUGH MPE FILE
                           WITH DATA MGMT VERB
     3.000  0000
     3.100  0000        DEFINE(ITEM) COUNT I(9,0,4);
     3.200  0000        LIST COUNT,INIT;
     4.000  0002        LIST RECORD,INIT:TIMESTART,PROCTIME;
     5.000  0007        WHILE (COUNT)<200
     6.000  0007           DO
     7.000  0007 1         GET(SERIAL) MPEFILE,LIST=(RECORD),STATUS,
                                       NOMATCH;
     8.000  0013 1         LET (COUNT)=(COUNT)+1;
     9.000  0016 1         DOEND;
    10.000  0018        LIST TIMESTOP,PROCTIME:ELAPTIME;
    11.000  0022        LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    12.000  0024        DISPLAY ELAPTIME;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:03
   ELAPSED TIME=00:00:03

*******************************************************************

TRANSACT/3000    HP32247A.02.02 - (C) Hewlett-Packard Co. 1984

SYSTEM NAME>

ELAPSED TIME
 4615

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T23

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000            SYSTEM T23,file=MPEFILE(READ(OLD));
     2.000  0000      << PROGRAM TO TRAVERSE THROUGH MPE FILE
                         WITH FILE  VERBS>>
     3.000  0000
     3.100  0000      DEFINE(ITEM) COUNT I(9,0,4);
     3.200  0000      LIST COUNT,INIT;
     4.000  0002      LIST RECORD,INIT:TIMESTART,PROCTIME;
     5.000  0007      WHILE (COUNT)<200
     6.000  0007         DO
     7.000  0007 1       FILE(READ) MPEFILE,LIST=(RECORD);
     8.000  0014 1       LET (COUNT)=(COUNT)+1;
     9.000  0017 1       DOEND;
    10.000  0019      LIST TIMESTOP,PROCTIME:ELAPTIME;
    11.000  0023      LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    12.000  0025      DISPLAY ELAPTIME;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:03
   ELAPSED TIME=00:00:03

*****************************************************************

TRANSACT/3000    HP32247A.02.02 - (C) Hewlett-Packard Co. 1984

SYSTEM NAME>

ELAPSED TIME
 3715

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T24

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000           SYSTEM T24,file=MPEFILE(R/W(OLD));
     2.000  0000     << PROGRAM TO PUT INTO MPE FILE WITH DATA
                        MGMT VERBS>>
     3.000  0000
     3.100  0000     DEFINE(ITEM) COUNT I(9,0,4);
     3.200  0000     LIST COUNT,INIT;
     4.000  0002     LIST RECORD,INIT:TIMESTART,PROCTIME;
     5.000  0007     WHILE (COUNT)<200
     6.000  0007        DO
     7.000  0007 1      PUT MPEFILE,LIST=(RECORD);
     8.000  0013 1      LET (COUNT)=(COUNT)+1;
     9.000  0016 1      DOEND;
    10.000  0018     LIST TIMESTOP,PROCTIME:ELAPTIME;
    11.000  0022     LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    12.000  0024     DISPLAY ELAPTIME;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:03
   ELAPSED TIME=00:00:03

*****************************************************************

TRANSACT/3000    HP32247A.02.02 - (C) Hewlett-Packard Co. 1984

SYSTEM NAME>

ELAPSED TIME
 5139

EXIT/RESTART(E/R)?>
```

COMPILED LISTING AND RUN RESULTS OF PROGRAM T25

```
 COMPILING WITH OPTIONS: LIST,CODE,DICT,ERRS

     1.000             SYSTEM T25,file=MPEFILE(R/W(OLD));
     2.000  0000       << PROGRAM TO PUT INTO MPE FILE WITH FILE VERBS>>
     3.000  0000
     3.100  0000       DEFINE(ITEM) COUNT I(9,0,4);
     3.200  0000       LIST COUNT,INIT;
     4.000  0002       LIST RECORD,INIT:TIMESTART,PROCTIME;
     5.000  0007       WHILE (COUNT)<200
     6.000  0007          DO
     7.000  0007 1        FILE(WRITE) MPEFILE,LIST=(RECORD);
     8.000  0014 1        LET (COUNT)=(COUNT)+1;
     9.000  0017 1        DOEND;
    10.000  0019       LIST TIMESTOP,PROCTIME:ELAPTIME;
    11.000  0023       LET (ELAPTIME)=(TIMESTOP)-(TIMESTART);
    12.000  0025       DISPLAY ELAPTIME;

 CODE FILE STATUS: REPLACED


 0 COMPILATION ERRORS
 PROCESSOR TIME=00:00:04
   ELAPSED TIME=00:00:03

****************************************************************

TRANSACT/3000    HP32247A.02.02 - (C) Hewlett-Packard Co. 1984

SYSTEM NAME>

ELAPSED TIME
 4220

EXIT/RESTART(E/R)?>
```

3044. Software Design: Building Flexibility

Victoria A. Shoemaker
Michell Humphrey
2029 Woodland Parkway
St. Louis, Missouri 63146

I. Introduction

Three years ago, I was reading through a scope and evaluation report for a computer solution to the age old problem of tracking data through the corporate maze and came to the 'solution' statement which read something like this, 'It is apparent that this company needs a total intergrated interactive system to track....'. Wow! I was sure that I wanted to get involved in that project. How could I pass up the opportunity to get involved in a TOTAL INTERGRATED INTERACTIVE computer solution? Well, fortunately or unfortunately, I did get involved in this project. For those of you still in a haze of what the 'real' problem and solution was, let me explain.

The company had a need to record and report financial data through five different organizations. The data, once recorded, needed to be reported differently for each organization, as well as reported in some consistent format for company level reporting. Designing a system which would satisfy all of the requirements would be difficult and expensive, if not impossible.

In this paper, I would like to share some of the solutions that the project team came up with and some solutions that I have thought of via that useful design tool, hindsight.

II. Problem Statement

A discussion of the real problems and the situations which molded the requirements for the solutions may be helpful prior to getting into the technical merits and implementations of the solutions.

o Client Organization

The client was organized along product and functional lines. Within each functional area there were teams assigned to handling the needs of a particular product line. Budgeting, recording and reporting were done on both the functional and product level.

o Basic Reporting Needs

The client needed to provide summary reports for the entire company on both the product level and organizational level. Therefore, it was imperative that all organization collect and report a core of information to make this task feasible. In addition, each organization need to report summary information on the different projects for the management to evaluate the success or failure of each project separately.

o System Interfaces

The system needed to interface with a larger corporate financial system, thus reporting the financial activities of this major aggregate. The system that we were to interface had a fixed format and included many pieces of information that were not actively used by the client.

o Miscellaneous Headaches

The client, as a company, was a young and changing company. Their management members changed frequently, as did their procedures. There were many charismatic management members which provided the project with a great deal of energy, not all of which was directed in the same direction. All of the needs of the charismatic management needed to be met. To put it bluntly, not only did the system need to be able to stand the test of time, but acts of gods and demi-gods. It was evident, that if a successful system was possible at all, the key to its success would be flexibility.

III. Design Methodology

The project team was a little baffled as to where to start with this system. All of our 'normal' design methodology did not seem able to cope with the need to record the needs of and design a system which satified all of the needs of our client. It seemed to the project team that we may be dealing with five systems, not one. The needs seemed, at first, second and third glance, to be too different to try and integrate into one system. The major design problem became how does one intergrate five different coding methods, five different and changing reporting needs, and still meet the overall reporting requirements of integrate reporting on the company level, as well as be able to interface with the corporate financial system.

We elected to proceed through requirements definition as though we were collecting requirements for five different systems. This allowed us the flexibility of listening to each of the organization's needs without the pressure of trying to force fit their needs into an overall system definition. The results were fantastic. Each of the organizations felt as though they were the driving force behind the project and that we understood their needs the best. The ability to record all of their needs as though they were the only needs, allowed the client to organize their priorities prior to having to consolidate the overall project priorities. When it did come time to eliminate certain facets of their requirements, they had already gone through the distillation process of what is important and necessary and what is fantasy better left for another project. The consolidation process of overall project requirements went fairly well because everyone knew what they were willing to let go during the bargaining process and what they were not willing to sacrafice.

During the requirements definition process, just to give you a feel for the magnitude of needs of the different organizations, we collected 150 report definitions, 300 different data requirements, and 5 different views of system flow. The reporting requirements were all different, yet you could feel a commonality of purpose, but only at a general level- nothing that would keep you from writing 150 reports. It was at this point that we started investigating tools that would allow us to develop one system that would satisfy all of these different needs.

At the end of requirements definition, it became clear that we were developing a core system that satisfied the common data requirements, corporate interface and company-wide reporting needs. In addition to the core system, we needed to devise some customization techniques to satisfy the unique requirements of each organization. A quick review of the requirements of each of the parts of the system might be helpful. The core system needed to provide the following:

- o interface to the corporate system
- o the ability to summarize, for the entire company, all financial information from each of the organizations
- o satisfy the reporting requirements at the company level.

The customization features would have to fulfill the following requirements:

- o Data collection
  - ability to collect organizational specific data
  - ability to change coding structures as needs changed
- o Data entry
  - coding of data
  - error messages
  - on-line help screens
- o Reporting
  - ability to format and report their own customized data and common system data
- o Training Needs
  - ability to 'design' their own training session with the use of on-line training facilities
  - ability to have customized help screens during data entry
  - ability to have customized error and informational messages.

o General Design

Our overall approach of developing a core system with customization facilities was presented to the user and accepted. We then proceeded with the aurdous task of designing the major components of the system; namely, data collection, reporting needs, and training needs of the core system and customized features.

Data Collection

We used the data required by the corporate system and common data between the organizations as the data we would collect in our main data base. This data was to be put in a single data base with common data entry screens. The coding structures needs would be contained in a central control data base and read into the programs for the purposes of editing and reformating the account number. This scheme allowed the seperate coding structures without the need for seperate data entry screens. The codes were to be entered into a free form block, big enough to accommodate the largest coding structure, and then converted to the proper corporate format. The conversion process was to be parameter driven and totally independent of the code of the screens.

The other data that was special to each organization would be kept in a set of free format tables and would be defined via a data dictionary for each organization. If the data needs of the organization would change then one could merely change the dictionary to reflect these changes. The data could therefore be independent of any physical structure through the use of a dictionary.

Reporting Needs

The only formal reports that would be developed would be the company level reports. They would be provided with the system. All other reports would be written via a simple user report writer using the dictionary as its mean of pulling both custom and common data. Assistance would be provided in writing these reports, but the burden of writing most of reports would fall upon the user. Because of this approach to reporting needs, it became apparent that there was a great need need for adequate and flexible training.

Training Needs

The problems of developing a training methodology which took into account the changing personnel as well as the customized features were numerous. The normal approach of developing an operator manual was not satisfactory. The other approach of custom training sessions seemed too time consuming and not very practical. It was decided that the system would have to be 'self-documenting'. I know, every system is self-documenting. But, alas we were truely faced with developing the first, and maybe only, self-documenting system. The approach we elected, was the use of help screens, developed by the user, and a simple tutorial program. The tutorial program presented small amounts of factual information and then quizzed the user about the material.

IV. Design Tools

Now that the requirements and general system approach are clear, a discussion of the techniques used to accomplish the overall design is appropriate. The techniques that we used are technically simplistic and the implementation of them is not difficult.

Modular Code

I know that we all conceptually know that developing code in small, well defined modules is correct, yet we all seem to develop code which satisfies the initial requirements, but is not as easy to maintain as one would like. These sacrafices are normally done in the interest of saving time. Well, I too have fallen into this evil temptation, but now that I am working for a software house, I have seen the error of my ways. If there was any one area that we could have improved upon, it would have been this area. We succumbed to pressure and developed code which later became difficult to change without some significant structural changes. Please listen and repent, write code in small, well defined modules and you will be saved the software god's wrath and punishment of spending late evenings doing rewrites.

Data Dictionary

The data dictionary allowed multiple views of the common data and the definition of data specific to a particular organization. Being that all of the reports were driven by the dictionary, commonality of reporting methods were maintained. The dictionary, in addition to providing fixed view of the common data, allowed the user to merge their specific data with the common data. The dictionary chosen was easy to modify and provided simplistic security features. These security features ensured the client of their data privacy.

Customized Tables

This tool, together with the data dictionary, allowed us to meet the customization requirements. The technique we used was very simple. We developed a data base that contain tables of various lengths. The keys to these tables were the table name and the key value. The users used the dictionary to define the contents of each of table. As the user's data needs changed, they could change the definition of the tables in the dictionary.

Message Catalogs

As of MPE IV, message catalogs became available to the common user. A message catalog is a collection of messages keyed by a message number. These catalogs are built via the MAKECAT program available in PUB.SYS. The obvious advantage of using a message catalog is not having to hard code each of the messages in your program. It also allows you to customize your messages for each client, including writing the messages in different languages. Another advantage of message catalogs is that they are allocated to a system extra data segment which provides access to your message at a speed akin to greased lightning. We set up a message catalog for each of users, which they were able to customize to their needs. This customization required little or no effort on the data processing staff. The catalogs could be changed at any time and as often as the user required.

Help Facility

This facility at first seemed like the hardest to develop. What we ended up with is an elegantly, simple approach. We developed a data base that was keyed by userid and screen id. The only data in the data set was a name of a file which contain helpful hints written by the user. When the user hit the help button, the program would fetch the file name from the data base. The program would then take the terminal out of block mode and display the text. When the user was 'finished', the terminal would be placed back into block mode and the screen redisplayed. I was surprized by what the user put into these files. The files contained common codes, examples of reports, people to contact in case of problems, and manual/automatic procedures.

Tutorial System

This tutorial system was another simple solution to our training problem. What we wanted to get around was having to write a training system for each organization's customized subsystem. What was developed was a data base with two types of data; text or factual information and the question and answer data. The text data was keyed by topic and organization. The question and answer data was keyed by topic. There were a number of questions. The questions were randomly selected and the answers were randomly displayed. The answer correct answer was always displayed. When the user selected the correct answer, he was given an appropriate message of praise. If an incorrect answer was chosen, he was prompted again for the answer. Score was kept and displayed to the user at the end of their session. The users developed both the factual information and the question and answer information. The little subsystem was later used in other systems.

Control Database

The control data base was our answer to the different coding structures. The data base contained a 'key' to their coding structure. The 'key' is read in by the program, as its guide to the data requirements. The users had the option of using their custom tables to edit these codes or wait until they were edited against the corporate system.

Profileable Processing

This is one of those tools developed via that useful design tool called hindsight. In the original design, all of the conversion that took place between the organization's code structure and the corporates, took place in a custom program. This program converted each organization's data and output a file to corporate system. It has occurred to me that these conversion routines could have been written in the report writer using custom tables for the conversion. This would allow the user to change his profile as he desired.

It has also occurred to me that the data dictionary could have had more 'profileable' options for editing on the screens. The organization's 'profile' could be read into the data entry screen program at the beginning and could control what processing went on. My feeling about this particular feature is that, although feasible, it may not have been easily justified for a single purpose system.

V. Conclusions

It is in the conclusion of the paper that one tries to drive the the point home in a clear and concise format. If I were to try to make one point via this paper it would be, it is possible to design a flexible, yet simple system that does similiar processing. Although this type of system is feasible, it will only be cost worthy if you decide upon this approach from the beginning of the design process. It will allow you to develop a system which is easy to maintain and enhance. It will save you from going back and having to retrofit a system to a 'new' organization's needs. The key points of the approach we have discussed are:

- Pre-planning
  Determine what features that you would like to implement. Know what is going to be included in your core system and what features are going to be customizable.
- User Support
  It was obvious that the burden of a heavily customized system is on the user. If your user is not very sophisticated or active in the design process, then stay away from this approach. A customized system will only work if the user is active and comfortable with the tools that you are going to provide them. This approach is not for a user who does not know what a computer is, much less how to logon.
- Do not build Rome in one day
  It is not humanly possible, regardless of what kind of superstar you are, to implement all of the features, both custom and core, in one day. Implement your features in modules. Bring one module on at a time. Let the dust settle before attempting to implement another module.
- Carefully determine what your core system will be.
  It is real easy to include things in your core system which are not truely shared by all your users. If your core is not pure, you find that implementing your customized features will become difficult. If you find that your core is very small, then maybe you do not have a system that will fit this approach. You may have to finally admit that you are really dealing with five different systems.
- Choose your development tools carefully
  There are a number of data dictionaries and fourth generation languages on the market. Many of these packages are excellent, but alas there are a number which are cumbersome and may not fit your overall objective of flexibility and ease. Evaluate the tools you plan to use and make sure they do not make more work than they save.
- Modular Code
  I know I have harped on this before, but this could be a universal truth and I don't want it to slip past. In any system, the more compact and modular your code is, the easier it is to maintain. I don't remember the numbers exactly, but I do remember that the majority of software dollars is spent in maintenance of existing systems. Make it easy on yourself or those that will end up supporting your code, write modular code. It is easier to debug when there are problems and easier to enhance when the need arises.
- Message Catalogs/Help Facilities
  Since the design of this system, MPE has allowed users the ability to add to the help facility that is part of MPE. This facility provides you, the analyst, with a very inexpensive help system. Take advantage of it. Its a freebie. They don't come around very often.

As with all system design methodologies, this methodology does not fit every business problem. This technique has been helpful for me for business problems which span multiple organizations, yet do essentially the same processing. The technique has some real disadvantages that one needs to be aware of namely:

- o Need for user support
  This approach will not work at all if the user is not behind the concept in more than verbal consent. The user plays a significant role in making the customization features work.
- o Need to decide on this approach from the beginning.
  If you do not elect to use this approach from the beginning you will find it next to impossible to retrofit once the design process has progressed significantly.

I hope this discussion of an approach to handling the problem I discussed, as well as a description of the techniques used, has been helpful to you. I would like to thank those people who have helped me put this paper together; Richard Diehl, Ed McLaughlin, Jim Kramer, and Mitchell Humphrey.

3045. The Twilight Zone ....Between MPE Capabilities.

Jelle Grim
Holland House
AALST, Holland.

Abstract.

Within the MPE Operating System a number of so-called Capabilities are recognized which define the actions a user (or a program) can perform on the HP3000 system. These capabilities can be used to implement and maintain the security and the integrity of the system. The most critical capabilities are SM (System Manager), OP (Operator/Supervisor) and PM (Privileged Mode).

The traditional HP3000 systems always had assigned to them a System Manager who, by means of his so-called Capability Set, had access to all resources of the system. However, with the introduction of the lower-cost HP3000 systems and the ongoing hardware decentralization, it will not always be desirable and/or feasible to let a full-time user of a decentralized system have all the capabilities. In other words : These systems will not have a full-time system manager and/or console operator.

This paper will describe a method, that can be used to assign capabilities to certain users dynamically instead of statically, i.e. only those capabilities and only for so long as is necessary to perform a specific task. Needless to say that this must be accomplished while retaining security and including all required measures to avoid undesirable adventures. As the "carrier" of this method a menu processor can be used, whilst also some intrinsic substitution will have to take place. These items will be discussed also.

Contents.

1. Introduction.

This presentation will deal with the requirements for a more dynamic definition of the so-called "capabilities", that are recognized within the MPE operating system of the HP3000 series of computers. Currently these capabilities are supported in a (too) static way. If we go back in the history of the HP3000 series of computers, we will find the cause of this situation.

When the first types of computers (the CX, the II, III, 44 and 64) of the HP3000 series were introduced and sold, the investment for a complete installation normally exceeded $150.000. The machines were used in a data processing environment, whereby a number of special functions were recognized. In connection with the HP3000 machines the function of System Manager, Operator and often also Account Manager were special and mostly required some extra training or the hiring of specialized personnel. In comparison with the investments in hardware and system software these specializations were justified.

The MPE operating system also recognizes these special functions by defining special capabilities that were assigned to these functions, enabling them to perform their specific, specialized tasks. The capabilities referred here are SM, OP and AM. Within the MPE security these capabilities give the user a number of special commands, like ALTACCT (SM only), ALLOCATE (OP only) and ALTUSER (AM or SM), and widen the scope of a number of other commands, like REPORT (especially SM and AM) and STORE (especially OP).

Lately Hewlett-Packard has introduced more and more low cost HP3000 systems, like the 39 and recently the 37. Currently it is possible to acquire a complete HP3000 system for less than $30.000. Now the picture changes, because it is no longer feasible to appoint expensive System Managers and/or operators full time for these small installations. In the best case, if these small installations are part of a multi-machine user, a specialist of the central DP department will dedicate some time to these mini-mini's now and then, thereby acting as a part-time or ad hoc System Manager.

However, some tasks involving special capabilities have to be done on a regular basis or have to be done on an ad hoc basis, when a specialist is not available. This can be solved by assigning one or more of these special capabilities to one or more of the regular users of these little systems. Now here's the catch. The special capabilities OP and even to a greater extend SM give the user, who has these capabilities assigned to him, almost unlimited power within the MPE operating system and its security. Obviously this situation is not very nice, either because of educational reasons (playing system manager or operator just requires some specialist knowledge), or because of security reasons (often it is not advisable to let a "regular" user have access to all information stored within a machine).

Currently there is no "in-between", i.c. there is no such thing as a sort-of System Manager or a partly Operator or a pseudo Account Manager, in other words :

There is no twilight zone between MPE capabilities.

Thinking of this problem, it seems that a lot of things will be solved, if we would have a mechanism for dynamic capability switching. Dynamic Capability Switching, in this context, would mean, that a "normal" user gets a special capability (OP, SM, AM, etc.) assigned to him for so long a time as he needs it to perform a certain task. After that task is finished, he has to return to the status of "normal" user again. Furthermore, once we would have this mechanism we could also use it for the following, additional operations that we have been thinking about, but that were impossible to implement under the standard MPE operating system :

- A user with temporary AM capabilities could be allowed to change his own password, to perform some file management within his own account, BUT he must NOT be allowed to perform a PURGEUSER or PURGEGROUP command.

- A user with temporary OP capabilities could be allowed to use commands like ALLOCATE or SWITCHLOG, to alter the JOBPRI to be able to submit a job to the CS queue, BUT he must NOT be allowed to perform a STORE command for files outside the logon group.

And, of course, once the problem of dynamic capability switching is licked, it will also be applicable to other capabilities like SF (save files), PH (process handling), DS (extra data segments) and PM (Privileged Mode !!!).

Although it seems that the only thing, that is needed now, is some sort of capability switching utility, there is more to this alteration of the MPE capability principle than meets the eye.

Therefore, this paper will elaborate on a number of design considerations with regard to the Dynamic Capability Switch, with rough designs of the components that, together, will form a complete dynamic capability switching mechanism. The concluding chapter will describe a way to integrate all components as described above into one unit, whereby an applications manager will be the end result.

2. Design Considerations.

In this chapter we will try our hand on describing a number of features and other assorted items with regard to the design of a dynamic capability switching system. As with most utility-type software, it is possible to turn out a quick-and-dirty routine, that performs the capability switching, without looking at the implications on system security, system integrity, etc.

However, thinking about the potential of a capability switching utility, everyone will agree that at least some caution will have to be applied when designing, building and using this software. This paper will not try to present an all watertight design for the capability switch, but it will merely point out a number of pitfalls, temptations and dangers, that have to be avoided at all costs.

In the following pages a number of items with regard to the design of the dynamic capability switch will be discussed.

Stand-alone or Routine.

The choice that has to be made here is between supporting the capability switch as a stand-alone program and supporting the switch as a routine/procedure. Defining it as a program enables the usage of the switch from within a UDC. On the other hand, when the capability switch is designed as a routine, it can also be used from within programs. If the switch must be used from a menu processor either method will suffice, because most menu processors support either the execution of programs, or the execution of external routines or both.

Simple Operation.

It seems best to keep the capability switch as simple as possible. The basic function of the capability switch must consist of :

- the ability to switch one or more MPE capabilities on,
- the ability to switch one or more MPE capabilities off,
- the ability to report on the current setting of the MPE capabilities, and
- any combination of the above.

The actual capability switching will be done on the MPE Job Information Table (JIT). No additional functions should be incorporated in the switch. The specification of capabilities that have to be switched ON or OFF and/or the command needed to verify the settings can be passed to the program or routine using a character string.

Usage of Privileged Mode.

It is obvious, that it is necessary for DCAPS, to be able to perform its feats, to use the much feared Privileged Mode capability. Although the PM usage for this purpose is no direct danger to the MPE integrity, indirectly breaches of integrity/security can occur when the real hackers enter the game.

The most stringent security measures must therefore be observed when determining the location of the program, determining who is going to use the program and how the program will be used.

Access Security / Location.

As the capability switching utility can be a dangerous weapon in the hands of an end-user, or worse, in the hands of an evil minded end-user a number of precautions must be taken to ensure, that the capability switch can and will only be used in a controlled environment. It doesn't do to leave the possibility open for just anybody to say "please run DCAPS and give me all capabilities".

Therefore both the location and the access of the software must be controlled. If the switch will be developed as a program, the best place for it to reside is within a group with the PM capability within the SYS account, like PRIV.SYS or UTIL.SYS Furthermore, the program should be protected by a lockword. If the switch is developed as a routine in an SL, then this SL must reside in the same group as the program that calls it, for instance a menu processor. It could be placed in the system SL, but that would give too easy an access.

Restricted Usage.

When a user has certain special capabilities assigned to him it is no good, to tempt him by letting him have unlimited access to the MPE operating system. As discussed in the Introduction, special capabilities will allow the user to use certain special MPE commands and/or widen the scope of certain other MPE commands. It is therefore mandatory to use the switch always in combination with some kind of restrictive user interface like User Defined Commands (logon UDC's) or a menu processor like HPMENU, HELLO-3000 or UNIMAN, making it impossible for the user to abuse the power vested in him by the capability switch.

The Danger of the MPE COMMAND Intrinsic.

Even if a restrictive user interface is used, the structure of the MPE subsystems sometimes offer the possibility to access MPE commands in a more indirect way. Subsystems like SPOOK, FCOPY, EDITOR and also user designed software can offer access to the MPE commands via the so-called COMMAND Intrinsic. If the user has some additional capabilities assigned to him by the capability switch at that moment, he can use the MPE commands belonging to that capability.

A second security enforcer must therefore be implemented in the form of a restrictive command intrinsic, i.c. a custom made command intrinsic, that captures all calls to the "official" MPE COMMAND intrinsic and that only transmits commands, that cannot be used for less legal purposes, to the MPE COMMAND intrinsic. Again, to make the complete capability switching system work, a restrictive user interface is necessary.

The design considerations as discussed above show, that it is not good practice to just write a little PM program, that will perform your capability switching. On the contrary, after some thought anyone will agree that, to support a real dynamic capability switching system, the following components are necessary and should always be user together :

- The actual Dynamic Capability Switch, or DCAPS.

- The Restrictive Command Intrinsic, or RESCOM.

- The Restrictive User Interface, or MENUPROC.

The following chapters will deal with each of these components in detail.

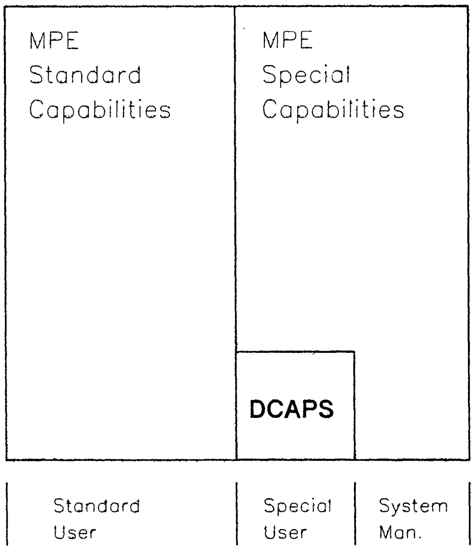
3. DCAPS, the Dynamic Capability Switch.

To get the logical placement of DCAPS into perspective, the diagram below shows how it can help a "special" user to access the MPE special capabilities.

Three types of users will be recognized :

- The standard user, who has normal access to all standard capabilities of the MPE operating system.
- The special user, who is just a normal user that now and then gets one or more special capabilities to perform a certain task
- The system management that normally has all capabilities at its disposal.

Figure 1. DCAPS, the Dynamic Capability Switch.



The Dynamic Capability Switch or DCAPS must be capable to perform at least the following activities :

- Switch one or more of the MPE capabilities ON.
- Switch one or more of the MPE capabilities OFF.

- Report on the current settings of the MPE capabilities (ON or OFF).

Some observations can be made. If DCAPS is envisaged as a stand-alone program, the actual capability switching will have to be performed on the MPE Job Information Table (JIT), which means, that the capability switch will be valid until it is switched off again or until the end of the current session/job.

It is therefore of utmost importance to keep track of which capabilities have been assigned and to switch the capabilities off when they are not needed anymore. The information required for the actual capability switching can be transmitted to the DCAPS program by means of the ;INFO= parameter of the :RUN command. The following rules can be implemented :

- If a capability is specified in the INFO-string either as it is or preceded by a plus sign (+) the capability will be switched ON.
- If a capability is specified in the INFO-string preceded by a minus sign the capability will be switched OFF.
- Multiple capabilities specified in the INFO-string must be separated by commas (,).
- The occurrence of the word VERIFY in the INFO-string, separated from the other information by commas, will cause DCAPS to display the status of the capabilities (ON or OFF) after all capability switches have taken place.

The following examples would be valid commands to run DCAPS according to the rules as laid out above :

```
:RUN DCAPS.PRIV.SYS;INFO="AM,PH"
:RUN DCAPS.PRIV.SYS;INFO="VERIFY"
:RUN DCAPS.PRIV.SYS;INFO="-SM,AM,+OP,+PH,VERIFY"
```

Another approach is to design DCAPS as a routine, residing in a privileged SL. For purposes of clarity, this routine will be viewed as accepting as a parameter one character string with the same layout as the INFO-strings as described above. A valid call to the routine would then be :

```
DECAPS (INFO); or
CALL DCAPS (INFO) or
CALL DCAPS USING INFO
```

whereby INFO would be a character string with the contents "AM,PH", "-SM,AM,+OP,+PH,VERIFY", etc.

An advantage of this method of parameter passing is, that to the user nothing changes in the case that Hewlett-Packard decides to add a number of new capabilities, change the effect of one or more capabilities, etc. Only DCAPS must then be changed to support the new (changed) possibilities.

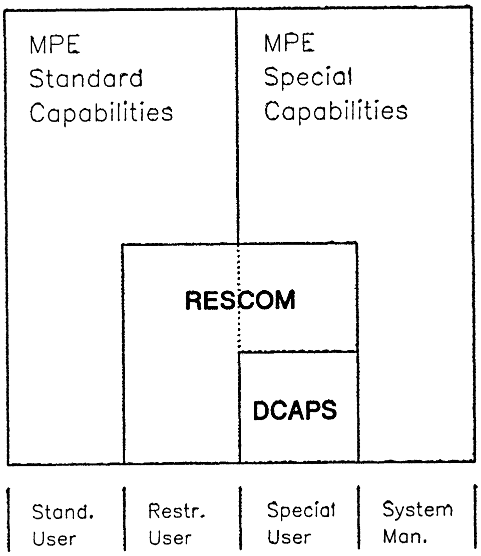
4. RESCOM, the Restrictive Command Intrinsic.

As can be seen in the diagram below, the RESCOM idea does not have to be restricted to working together with the DCAPS utility. RESCOM can also be used to restrict a "normal" users in the MPE commands, that can be executed programmatically.

Now four types of users can be recognized :

- The standard user, who has the standard MPE possibilities at his disposal.
- The restricted user, who can only perform programmatically the commands, that are accepted by RESCOM.
- The special user, who will now and then get permission to use one or more special capabilities, but who can only execute programmatically the commands, that are accepted by RESCOM.
- The System Management that normally has all capabilities at its disposal.

Figure 2. RESCOM, the Restrictive Command Intrinsic.



In order to intercept calls to the MPE command intrinsic, screen the contents of the call and to decide whether the call should be rejected or submitted to the MPE command intrinsic, an alternate command intrinsic, or RESCOM, must reside in the SL, that is attached to the program calling the MPE command intrinsic.

This alternate command intrinsic or RESCOM must recognize the following authorities, possessed by the user calling the intrinsic :

- The authority to execute all MPE commands.
- No authority to execute any MPE command.
- The authority to execute a number of MPE commands.

The first two possibilities will not present any real problems when designing RESCOM. The last one, however, is more tricky and it requires at least some kind of list of authorized or non-authorized commands. The choice must be made by storing these restrictions either in a file or in an extra data segment. For performance reasons the extra data segment method will be preferable.

An ideal situation would be if RESCOM could be used for both "normal" MPE commands and programmatically executed MPE commands. However, this excludes MPE itself (via its UDC structure) as the user interface. In this case a menu processor like HPMENU, MenuProcessor or UNIMAN must be used.

Just to illustrate how a menu processor can be used to build an alternate (restricted) MPE command processor the following will show a part of a UNIMAN command section doing just that. As the author is familiar with the UNIMAN menu processor, this package will be used in all examples pertaining to the menu processor. See the next chapters for some more information on UNIMAN.

```
*
* EXAMPLE COMMAND INTERPRETER FOR THE WASHINGTON CONFERENCE
*
COMMAND COM06,CONTROLLED
SET UPSHIFT
*
LABEL LOOP
DISPLAY :
ACCEPT MPE
IF PARM MPE EQ BYE
   STOP
ENDIF
IF PARM MPE EQ MENU
   GOTO END
ENDIF
:!MPE!
IF ERROR
   DISPLAY !ERROR!
   DISPLAY ENTER "MENU" TO GET BACK TO THE UNIMAN MENU
ENDIF
GOTO LOOP
*
LABEL END
END
*
*
```

Of course, when executing this file, RESCOM must be activated by running UNIMAN with the ;LIB= parameter, indicating the SL that contains the RESCOM routine. Furthermore, RESCOM must have the information on the commands that are authorized and/or the commands that are not authorized, at its disposal.

As soon as this part of the UNIMAN designer file is executed, the screen will be cleared and a normal MPE prompt (:) will be displayed. From this point onwards, any MPE command entered will be executed, if it is OK with RESCOM. The BYE command will terminate the execution of UNIMAN. If UNIMAN is part of a UDC ending with a BYE command, the user will be logged off automatically. If the command MENU is entered, control will be returned to the UNIMAN MENU-section, that called this COMMAND-section COM06.

Special considerations have to be taken into account when determining the location of the SL containing the RESCOM intrinsic. For all accounts outside the SYS account, that have to run programs to be restricted by RESCOM, the SL containing RESCOM must reside in the PUB group of that account and the programs must be run with the ;LIB=G or ;LIB=P parameter.

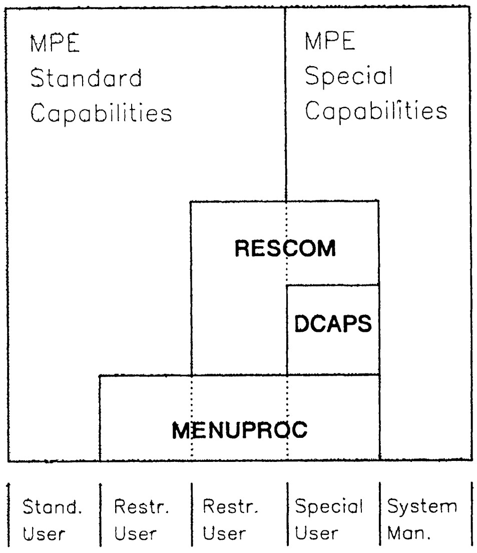
For programs in the group PUB.SYS, there is a catch, because PUB.SYS already contains an SL (the system SL). It is therefore necessary to migrate all programs in SYS, to be restricted by RESCOM, to a group like, for instance, UTIL.SYS or LIB.SYS. The SL containing RESCOM must also reside in that group and all programs must be run with the ;LIB=G parameter.

5. MENUPROC, the Alternative Command Interpreter.

Because DCAPS, the Dynamic Capability Switch, uses some special capabilities itself (especially the PM capability), it is mandatory that the regular user or even anybody not being the System Manager has no access to this program. Also the possibilities created using DCAPS should be shielded from direct access by the user. It is therefore, that some form of alternative command interpreter or menu processor should be used to restrict the activities of the users outside the application that they must process.

Enter MENUPROC. For the purpose of this presentation MENUPROC is not an existing system, but more a name to be given to the idea of having an intermediate layer between the user and the power of MPE. MENUPROC can be anything from a shrewd application of the UDC structure, using logon UDC's up to a more developed menu processor like HELLO-3000 or UNIMAN.

Figure 3. MENUPROC, the Universal Menu Processor



A MENUPROC system to be used for this purpose should be designed along the following reasonable criteria :

- MENUPROC should guide the user from the moment he logs on to the machine until he logs off again. It should be impossible for the user to access MPE directly. The logical way to solve this one is start MENUPROC from a logon, non-breakable UDC.

- MENUPROC must at least be capable of
  . Running programs.
  . Streaming jobs.
  . Executing MPE commands.
  . Detecting errors.

- It would be an advantage if MENUPROC would possess its own security layer, enabling, for instance, extra passwords for certain for certain sensitive actions.

- It would be an advantage if MENUPROC would have the possibility for the conditional execution of certain actions based on, for instance errors detected, the current logon device, the time of day, etc. Looking at this list it will be obvious that rather simple systems such as the MPE UDC's and HPMENU will have much difficulties in meeting the design criteria for MENUPROC. However, numerous menu processing systems are on the market or can even be found on the Interex Contributed Software Library. It is the responsibility of the system management to select and test one or more of these systems to be used in combination with DCAPS and RESCOM. It is not advisable to start building one's own MENUPROC because the market offers systems for any purpose at reasonable prices.

In order to see, how the combination of DCAPS, RESCOM and MENUPROC can work, the UNIMAN package will be used to fulfill the tasks of MENUPROC. More information on UNIMAN can be found in the UNIMAN User Manual. For the time being if suffices to say, that UNIMAN uses a designer file, consisting of MENU sections, describing screen layouts, the function key labels, and the names of the COMMAND sections, that have to be executed when a certain key is pressed, and COMMAND sections, describing the actions to be performed. The UNIMAN language is rather selfdescribing. In the following examples detailed explanations will be provided in those cases that are not directly clear on reading.

```
*
* DEMONSTRATION UNIMAN DESIGNER FILE FOR WASHINGTON CONFERENCE
*
* INITIALIZATION, UNIMAN PASSWORD CHECK
*
COMMAND INITIAL
CLEAR
DISPLAY AT 1010,PLEASE ENTER YOUR UNIMAN PASSWORD :
GETPASS EXPASS,CONSOLE
LOAD DEMO1
END
*
*
*
*
*
* SIMPLIFIED USER MENU DEFINITION
*
MENU DEMO1
DISPLAY AT 1010,DEMONSTRATION MENU 1
DISPLAY AT 1110,====================
KEY 1,CHANGE\PASSWORD,COM01
KEY 2,TDP\,COM02
KEY 3,FCOPY\,COM03
KEY 4,STREAM\CS QUEUE,COM04
KEY 8,EXIT
END
*
*
* CHANGE USER MPE PASSWORD USING AM CAPABILITY
*
COMMAND COM01
CLEAR
DISPLAY AT 1010,PLEASE ENTER NEW PASSWORD :
ACCEPT PASSWORD
:RUN DCAPS.PRIV.SYS;INFO="+AM"
:ALTUSER !USER!;PASS=!PASSWORD!
:RUN DCAPS.PRIV.SYS;INFO="-AM"
END
*
*
```

```
* TEXT AND DOCUMENT PROCESSOR USED WITH STANDARD CAPABILITIES
*
COMMAND COM02,CONTROLLED
:RUN TDP.PUB.SYS
IF ERROR
   DISPLAY UNABLE TO RUN TDP
   DISPLAY !ERROR!
ENDIF
END
*
* FCOPY USED FOR FILE MANAGEMENT WITH AM CAPABILITIES
* MAKE SURE RESCOM IS ACTIVATED
*
COMMAND COM03,CONTROLLED
:RUN DCAPS.PRIV.SYS;INFO="+AM"
:RUN FCOPY.UTIL.SYS;LIB=G
:RUN DCAPS.PRIV.SYS;INFO="-AM"
END
*
*
*
*
* STREAM A JOB IN THE CS QUEUE USING OP CAPABILITIES
*
COMMAND COM04,CONTROLLED
DISPLAY AT 1010,ENTER NAME OF JOBFILE :
ACCEPT JOBFILE
:RUN DCAPS.PRIV.SYS;INFO="+OP"
:JOBPRI CS
:STREAM !JOBFILE!
:JOBPRI DS
:RUN DCAPS.PRIV.SYS;INFO="-OP"
END
*
*
* EXIT UNIMAN, PREFERABLY FOLLOWED BY A BYE IN LOGON UDC
*
COMMAND COM08
STOP
END
*
*
```

Remarks.

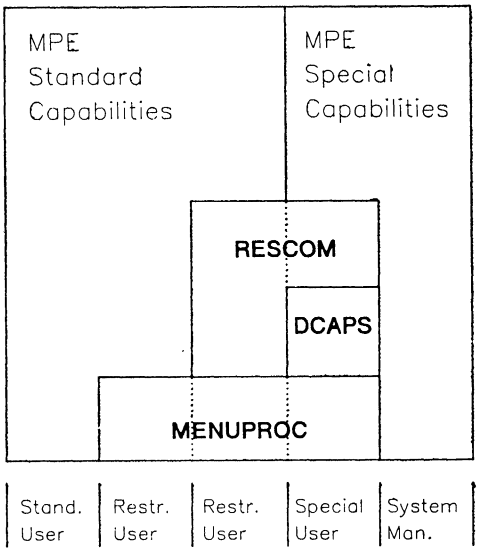
- The CONSOLE keyword of the GETPASS statement indicates, that all password violations must be logged onto the system console.
- The backslash (\) in the KEY statement causes the text in the function key labels to be centered.
- At the location where a parameter name is enclosed in exclamation marks (!), the value of that parameter is substituted.
- The keyword CONTROLLED of the COMMAND statement causes an automatic CLEAR and DISABLE at the start of the section and

a REFRESH at the end of the section.
- The UNIMAN parameter ERROR always contains the last MPE error encountered, if any.

6. Tying it all together.

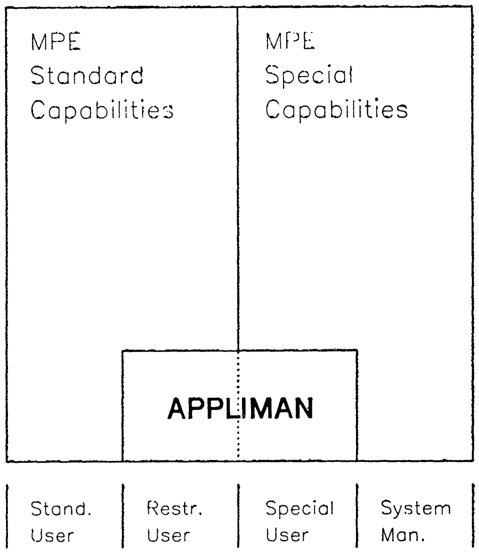
The three systems, that are defined so far, and their relations can be depicted by the following diagram :

Figure 4. The Components Needed for the Solution.



However, the three systems still are very separate and are just a number of tools to get to the goal as described in the introductory chapter. The last step is to integrate DCAPS and RESCOM into MENUPROC in order to get a real universal applications manager. The following diagram shows the new situation as viewed by the designers of the user menus.

Figure 5. The Integrated Applications Manager.



The thing to decide upon now is how to achieve this integration. Just combining the three programs and routines into one big program is not the solution, because that would require the complete program to run in Privileged Mode. No, the best thing to do would be to design a number of commands to add to MENUPROC, that can describe the designer's requirements and that interface to the DCAPS and RESCOM subsystems. Every MENUPROC system will have its own particulars, so this presentation will just limit itself to additions that will have to be made to UNIMAN in order to get something like APPLIMAN.

As the UNIMAN designer file must be compiled before it can be executed, a new compiler command $PM will be introduced. $PM indicates, that the designer file may contain statements requiring access to Privileged Mode routines, in this case DCAPS. Default will be $NOPM and the UNIMAN password file SKPASS will contain a list of users, authorized to use the $PM compiler command.

The next command to be added for usage within COMMAND-sections is SWITCHCAP, that will be used to switch capabilities. SWITCHCAP accepts as a parameter a character string as described in the chapter on DCAPS with the exception of the VERIFY keyword, so the following UNIMAN statements are perfectly valid :

```
SWITCHCAP AM,PH
SWITCHCAP -SM,AM,+OP,+PH
```

The UNIMAN VERIFY statement will be extended with the keyword CAP that will cause the display of the status (ON or OFF) of the MPE capabilities. VERIFY ALL will also include this display.

The integration of the restrictive command interpreter will be performed using an extra data segment to store all commands with their ON/OFF indicators. To initialize the status of the commands, two compiler commands have to be added : $ALLOW to initialize all commands as being allowed and $DISALLOW to initialize all commands as being disallowed.

The COMMAND-section statements ALLOW and DISALLOW must be added to allow or disallow commands on an individual basis. Both commands accept as a parameter one or more commands, separated by commas. An ALLOW for a specific command stays in force until it is disallowed by a DISALLOW command.

The UNIMAN VERIFY statement will be extended with the keyword ALLOW that will cause the display of the status (ALLOWed or DISALLOWed) of the MPE commands. VERIFY ALL will also include this display. The UNIMAN designer file as described in the chapter on MENUPROC will look like this after the addition of the new commands : (see next pages)

Using this setup, an extra security measure can be achieved. Instead of changing the capabilities in the Job Information Table (JIT), it is now possible to change most capabilities on the stack of the UNIMAN process. The advantage is that, when UNIMAN is aborted for whatever reason, the special capabilities will be gone also, so the user can never end up within MPE with some special capabilities left.

```
*
* DEMONSTRATION UNIMAN DESIGNER FILE FOR WASHINGTON CONFERENCE
*
$PM
$DISALLOW
*
* INITIALIZATION, UNIMAN PASSWORD CHECK
*
COMMAND INITIAL
CLEAR
DISPLAY AT 1010,PLEASE ENTER YOUR UNIMAN PASSWORD :
GETPASS EXPASS,CONSOLE
*
*
ALLOW BUILD,FILE,HELP,LISTF,PURGE,RELEASE,RENAME,REPORT,RESET
ALLOW RESTORE,SAVE,SECURE,SHOWJOB,SHOWME,SHOWOUT,SHOWTIME,STORE
ALLOW STREAM,TELL,TELLOP
*
*
LOAD DEMO1
END
*
*
* SIMPLIFIED USER MENU DEFINITION
*
MENU DEMO1
DISPLAY AT 1010,DEMONSTRATION MENU 2
DISPLAY AT 1110,===================
KEY 1,CHANGE\PASSWORD,COM01
KEY 2,TDP\,COM02
KEY 3,FCOPY\,COM03
KEY 4,STREAM\CS QUEUE,COM04
KEY 8,EXIT
END
*
*
* CHANGE USER MPE PASSWORD USING AM CAPABILITY
*
COMMAND COM01
CLEAR
DISPLAY AT 1010,PLEASE ENTER NEW PASSWORD :
ACCEPT PASSWORD
SWITCHCAP +AM
:ALTUSER !USER!;PASS=!PASSWORD!
SWITCHCAP -AM
END
*
*
*
*
* TEXT AND DOCUMENT PROCESSOR USED WITH STANDARD CAPABILITIES
```

```
*
COMMAND COM02,CONTROLLED
:RUN TDP.PUB.SYS
IF ERROR
   DISPLAY UNABLE TO RUN TDP
   DISPLAY !ERROR!
ENDIF
END
*
*
* FCOPY USED FOR FILE MANAGEMENT WITH AM CAPABILITIES
* MAKE SURE RESCOM IS ACTIVATED
*
COMMAND COM03,CONTROLLED
SWITCHCAP +AM
:RUN FCOPY.UTIL.SYS;LIB=G
SWITCHCAP -AM
END
*
*
* STREAM A JOB IN THE CS QUEUE USING OP CAPABILITIES
*
COMMAND COM04,CONTROLLED
DISPLAY AT 1010,ENTER NAME OF JOBFILE :
ACCEPT JOBFILE
SWITCHCAP +OP
:JOBPRI CS
:STREAM !JOBFILE!
:JOBPRI DS
SWITCHCAP -OP
END
*
*
* EXIT UNIMAN, PREFERABLY FOLLOWED BY A BYE IN LOGON UDC
*
COMMAND COM08
STOP
END
*
*
```

Biography.

Jelle Grim worked for the same company, the contractor Royal Boskalis Westminster from 1966 to April 1984. Starting as a civil engineer in the technical area he almost immediately switched over to the computer section. The Boskalis automation between 1968 and 1984 changed from in-house IBM S/3, through external data processing at a CDC service bureau using local Datapoint mini's, to in-house HP3000 equipment from 1978 onwards. When Jelle left Boskalis he was Information Network Manager in charge of a dual HP3000 network serving approximately 200 terminals and microcomputers both in Holland and abroad. In April 1984 Jelle and his partner Rene van Geesbergen together founded Holland

House, a company specializing in HP3000 system management consultancy and software products. Jelle is secretary of the Dutch Users Group HP3000 (DUG) and a member of the Amsterdam 1985 Host Committee.

3046. Store-And-Forward Data Transmission in a Multi-System Network

John P. Korb, CCP
Innovative Software Solutions, Inc.
10705 Colton St.
Fairfax, Virginia 22032

Most of us think of a network of HP 3000s as two or three or maybe even five to ten HP 3000s connected together with DSN/DS3000. We think of a network where DSCOPY, the P-to-P intrinsics, Remote File Access (RFA), or perhaps Remote Database Access are used to pass data between systems, with job streams, UDCs, and/or path-specific programs controlling the operations.

In most of these networks, each data path is treated differently, often because some paths have direct data source to data destination links, while other paths may have to cross one or two or more intermediate systems. This path specific "coding" of job streams, UDCs, and/or programs is acceptable for small networks with few paths, but presents a "design and maintenance nightmare" when large networks of fifty or more systems with tens or hundreds of paths are involved.

This paper presents one approach to providing a standardized interface to the application programmer to be used for transfering data from any point in a network to any other point in the same network utilizing a store-and-forward design philosophy.

Store-and-forward was chosen because of the realities of communicating between approximately 55 HP 3000 systems in many different time zones all over the world. With systems in many different time zones, each operating on local time, there is almost always a "nightly" backup going on on at least one of the systems. Without a store-and-forward philosophy, applications would have to be "smart" enough to take into consideration the time zones of the processors between the local processor and the data destination processor, the dump times of the "bridge" processors, etc. and might be confined to limited time windows for transmission.

Store-and-forward eliminates these worries from the application programmer/designer. No longer does a whole day's transactions need to be batched until some 2 hour time window. No longer are there the panic calls at 6 AM because one on the DS lines along the way was down, so nothing was transmitted, and no new attempt can be made until the next transmission window some hours away.

By adopting a store-and-forward network design, applications programmers can have their programs write transactions to the Network as they occur, and the Network will transmit the transactions as the necessary DS lines become available. If the transactions need to go to a central system some four or five DS lines away and one or more of the systems along the way are unavailable, there is no problem. The Network transfers the transactions as far as possible, stopping at the break in connection. When the connection is re-established, the transactions continue along on their way - all without the the programmer having to worry.

What is the Network?

The Network is a collection of programs, procedures, files, databases, and job streams which when properly configured provide a data transportation system with the capability of transporting data from any point in the Network to any other point in the Network.

What is its Purpose?

The purpose of the Network is to provide a consistent, reliable, standardized method of transferring data between applications programs on any HP 3000 CPU in the Network.

What are its Features?

- It can accomodate a configuration of up to 1024 HP 3000 CPUs.
- Each CPU can be configured to contain up to 15 "logical nodes".
- Each "logical node" is referenced by the application programmer via a 4 character PSD (Processing System Designator) code.
- Each "logical node" can have up to 32768 application "function" codes.
- Each "function" code can have up to 32768 application "process" codes.
- Up to 32767 pre-defined ciphers can be used for encoding transmitted data.
- The Network is based on store-and-forward operation, buffering data to disc when physical communications lines are not available, and when the application on the receiving end is not running.
- Each time an application program receives a data packet from the Network it also receives the "logical node", "function", and "process" codes of the application which transmitted the data packet.
- All application program access to the Network is via eight standard network Procedures.
- Up to 800 words (1600 bytes) may be transferred with one procedure call (ie. in one data packet).
- An application can provide a data record "type" parameter to the receiving application along with but not included within the data record.
- An application can provide a "heading" and "heading type" to the receiving application along with but not included within the data record.
- Depending on configuration, each Network user can be required to provide a unique Network password.
- When reading from the Network, the receiving application has the option of being placed on wait indefinitely if there is no data available, or waiting for an application program determined interval (1 to 255 seconds) for data, then timing

out and returning a "no data available" error to the application.

The Basic Components of the Network

The Network Software consists of three major modules: 1) The user interface (the Network Procedures). 2) The packet switching and transmission code. 3) The maintenance and utility code. The user interface provides eight procedures which the applications programmer uses to write data to or read data from the network.

The packet switching and transmission consists of three programs. The Traffic Control Supervisor Program (TCSP), which is run from a batch job and acts as the creator and controller of the packet switching program, and the packet transmitting program.

The Traffic Control Progam (TCP) performs the packet switching function, reading packets from its input queue and writing them to areas for local NETREADs or to the input queue of a packet transmitting program if the packet is bound for a remote system.

The Network Message Transmitter Program (NMTP) performs the data transmission function. One NMTP runs for each DS line configured. The NMTP opens a DS line to the adjacent system, sets up a file equation to the input queue of the remote TCP. It then reads packets from its input queue and writes them to the input queue of the remote TCP. By having one NMTP per DS line and separate input queues for each NMTP, a DS line "hanging" or an adjacent system being down or otherwise unavailable does not affect transmissions to other adjacent systems.

The Network maintenance and utility code consists of a set of programs used to:

1) Build an initial configuration of the Network on a processor.

2) Provide a means of modifying the configuration of the Network on a processor.

3) Provide a means of adding to, deleting from, or modifying the local configuration of the Network pertaining to which users may access the Network, what passwords they must supply when opening the Network, etc.

4) Provide reporting on the activity of the Network, including usage statistics by user.

5) Recover from system failures or other interruptions which prevent the TCSP and its child processes (TCP and NMT's) from closing their files, emptying their extra data segments, etc. and terminating normally.

Environmental Requirements

Because of the nature of the software, all of the Network Software is written is SPL. The code requires PM (Privileged Mode), MR (Multiple Resource Identification Number), PH (Process Handling), and DS (Extra Data Segment) capabilities, and to a limited sense is operating system specific (MPE IV, MPE V/P, and MPE V/E are all supported by one, common version of the software through specific routines for specific levels of MPE). The Network Procedures reside in two user-callable privileged system code segments, allowing the procedures to utilize Privileged Mode without the application programmer having to have PM or prep the application programs with PM capability.

User programs wishing to call the Network Procedures must have MR capability. While only a very small amount of code would be needed to remove this requirement, it was decided to not "do the user the favor" of obtaining MR, using it, then giving it up, in order to force the application programmer to have MR capability, and thus, further restrict who can access the Network.

How does one use the Network?

The Network is accessed through eight Network Procedures. These procedures are standardized and are the same on all HP 3000 systems in the Network. Using these procedures, one opens the Network, then reads data from or writes data to the Network, and when done, closes the Network. The Network Procedures set up and maintain the extra data segments(s), buffers, data base, and files necessary for interaction with the Network. Please take a moment to briefly scan the procedure descriptions in appendix A and the glossary before continuing.

The Network Procedures

As mentioned earlier, access to the Network is through the Network Procedures. These eight procedures perform the functions of opening, closing, reading from, writing to, controlling, explaining error messages, and retrieving information about the status of the Network.

Below is a brief description of each of the eight Network Procedures. For parameter information on the procedures, please see appendix A.

NETOPEN
NETOPEN is used by an application program to gain access to the Network. Before any data can be written to or read from the Network, the Network must be opened via NETOPEN. NETOPEN identifies the user to the Network, sets the source PSD code, SFUNC, and SPROC the Network will use to identify the source of any data packets transmitted by the user, and determines the cipher (if any) that will be used to encode any data packets transmitted by the user.

NETCLOSE
NETCLOSE is used by an application program to terminate access to the Network. When an application is finished using the Network, it MUST CALL NETCLOSE to gracefully terminate access to the Network. IF NETCLOSE IS NOT CALLED BEFORE THE PROGRAM TERMINATES, THE LAST RECORD RECEIVED FROM THE NETWORK MAY BE REPEATED THE NEXT TIME THE NETWORK IS OPENED WITH THE SAME PSD, SFUNC, AND SPROC (this depends on the local Network configuration). Note that NETCLOSE mode 2 gracefully terminates access to the Network, yet leaves the last record received in a state where it will be the first record read the next time the Network is opened (depending upon the local Network configuration). NETMODE 2 is provided as a means of avoiding the loss of a record which the application could not process (ex. when reading from the Network and writing to an output file, the application reaches end-of-file on the output file, and cannot process the record just read from the Network).

NETREAD
NETREAD is used by an application program to receive data from the Network. NETREAD returns: 1) the data record (packet), 2) the PSD code of the sender, 3) the SFUNC of the sender, 4) the SPROC of the sender, 5) the record type supplied by the sender, 6) the heading type supplied by the sender, and 7) the heading data supplied by the sender.

NETWRITE
NETWRITE is used by the application program to transmit data (pass data to the Network). The application program provides NETWRITE with: 1) the data record (packet), 2) the PSD code of the destination, 3) the DFUNC of the receiver at the destination, 4) the DPROC of the receiver at the destination, 5) the record type of the data record (packet), 6) any heading data for the data record (packet), and 7) the heading type for the heading data. The record type of the data record, the heading, and the heading type are required by the Network to be present, but can be set to zero if the application has no need for them. The DFUNC and DPROC are used to determine which application at the destination the data is to go to. The users of the Network should coordinate between themselves which values each application will use to avoid conflicts.

NETCONTROL
NETCONTROL allows the application program to change the values used as the source PSD, SFUNC, SPROC, and CIPHER. NETCONTROL saves the effort (and overhead) of having to NETCLOSE, then NETOPEN with new parameters.

NETEXPLAIN
NETEXPLAIN is generally called by the application program after an error has been returned by one of the Network Procedures. NETEXPLAIN displays an error message, and performs some internal Network cleanup after an error. The application has the option of having NETEXPLAIN display the error message on the $STDLIST device, or returning the message in a buffer for printing by the application itself.

NETINFO
NETINFO allows the application to determine which lines are connected

and other information about local Network activity. The information provided by NETINFO is only as recent as the last NETSTATUS call.

NETSTATUS
NETSTATUS provides the application with up-to-the-minute status information pertaining to local Network activity.

PSD Codes (Processing System Designator Codes)

PSD codes are used by the Network to determine the source and destination of message packets. A PSD code is a 4 character code left justified and right filled with blanks if less than 4 characters in length. The PSD code is used only as an interface to the application - the Network Procedures perform a table look-up on PSD code to obtain the internal binary address which the Network uses for packet routing.

Because a table look-up is performed to translate the PSD code into a binary value, switching destination PSD codes frequently (ie. when sending via NETWRITE) is not advised. Where packets must be sent to more than one destination, system performance can be markedly improved by sending with as few destination PSD code changes as possible.

Multiple PSDs are allowed on each CPU. Each PSD code has its own unique internal binary code, and is addressed separately from the application program's perspective. PSDs residing on remote CPUs are addressed exactly as those on the local system, making transmissions within the local CPU no different from transmissions to remote systems once, twice, or many systems removed from the local system.

Because PSD codes are external references to the Network's internal binary addresses, and references to PSDs are table driven, relocating a PSD to a different CPU in the Network is possible, and in fact, relatively simple. Only the table references within the Network are modified - and only to the extent of indicating the new binary Network address for the PSD. This makes relocating applications across CPU boundaries a simple procedure that does not require any program recompilations or other changes.

Ciphers

The Network provides for up to 32767 ciphers. The ciphers are 16 bit values and are referenced by the application programmer by specifying a Cipher Number. The Network uses the Cipher Number to retrieve the Cipher Value from the Network's Cipher Table. This has two important implications.

First, since the application effectively provides an index into the Cipher Table, Cipher Values can be changed by the Network Administrator without requiring that the application software be changed in any way.

Second, since the actual cipher is kept in a table, anytime a cipher is added, changed, or deleted, every Cipher Table in the Network must be promptly and almost simultaneously updated to reflect the addition/change/deletion.

As with PSD codes, because the ciphers are kept in a table, the more the number of changes of Cipher Number, the more table look-ups the Network must perform, thereby degrading overall system performance. Use of a single cipher from NETOPEN through NETCLOSE is recommended.

User Blocking/Deblocking

As with any data transmission network, every packet transmitted requires a certain amount of "overhead" data. Since this overhead data is fairly constant in length, data transmission efficiency is maximized when medium to long packets are transmitted. Short packets result in a much greater overhead-to-data ratio, and thus, are relatively inefficient.

If you have multiple records of data to transmit, and each record is rather short (say 80 bytes), consider buffering up several records at a time into a block of multiple records, then passing the entire block to NETWRITE. This markedly improves data communications efficiency by: 1) reducing the amount of overhead that must be transmitted over the communications lines, and 2) reducing the number of packets which must be processed by the Network software (and thus, the number of disc I/O operations). Remember, the Network can handle records of up to 800 words in length (1600 bytes). If you are sending 80 byte records one-by-one, 20 packets (and the processing they require) are necessary to transmit 20 records. If you buffer the records into a block, only one packet is necessary.

Preliminary performance data suggests that it takes almost 20 times as long to transmit and receive 20 packets each containing 80 bytes of data as it takes to transmit 1 packet of 1600 bytes (these figures are based upon the transmission of a benchmark file of 1,281,360 bytes of data at 80 bytes per packet, at 800 bytes per packet, and at 1600 bytes per packet).

Due to the data independence of the "User Record Type", "User Heading Type", and "User Heading", these fields can easily be used to provide partial block information.

Long Records (greater than 800 words)

The Network can easily handle records of up to 800 words (1600 bytes) in length. Records longer than 800 words do present a problem. Below are some suggestions for handling records of more than 800 words.

If your record length lies in the range 801 to 960 words (1601 to 1920 bytes), you can transmit your record in one packet by utilizing up to 160 words in the "heading data" parameter of NETWRITE and NETREAD (960

= 800 + 160). To pass a record of between 801 and 960 words, pass part of your record as "heading data" (the HDATA parameter to NETWRITE), and the rest the "record data" (the RDATA paramter to NETWRITE).

If this method is used, be sure to pass NETWRITE the correct lengths for HDATA and RDATA in the HLEN and RLEN, and tag your record either with a special HTYPE or RTYPE value so the receiving application can recognize what you have done.

If your record length is greater than 960 words, you will have to split your record up into sections of 960 words or less. The Network software provides four (4) flags which you should use when splitting records. These flags are "Continued", "End-of-Record", "End-of-Block", and "End-of-File", and are passed in the "FLAGS" parameter of NETWRITE. THESE FLAGS HAVE NO SIGNIFICANCE TO THE NETWORK. Your application program on the receiving end must look for and interpret these flags and determine what operations to undertake.

Keep in mind that at the receiving end there is always the possibility that if two people were transmitting data to the same destination and one or both are performing multiple packet writes to the Network, the packets may be received interleaved. Thus, if you are expecting a lot of activity at a specific destination address (PSD, DFUNC, DPROC combination), your application must be able to handle interleaved transaction/file packets. The SRC, SFUNC, and SPROC parameters returned by NETREAD identify the source of the packet, and should allow a receiving application to properly reconstruct multiple interleaved packets. Additionally, if long, multi-packet transactions are being transmitted, using the "heading data" area to tag each component packet with the name of the file or the ID of the transaction might be employed to further prevent any intermixing of data.

Accessing the Network

Much like an iceberg, most of the Network is hidden from the application programmer. The application programmer deals only with the Network Procedures - the user interface to the Network.

The application programmer opens the Network, reads and/or writes from/to the Network, then closes the Network. To the application programmer the Network is accessed via Network Procedures, much as a database is accessed via IMAGE procedures. The parameters passed to/from the Network Procedures are always passed in the same order, although some parameters may be omitted from some procedures.

To prevent misrouting of data, only one user at at time may open the Network with a given PSD, SFUNC, and SPROC combination. Thus, if you desire to both transmit and receive at the same time, both functions must be performed by the same program. (Of course, many programs may have the Network open from a given PSD at one time, they just may not have duplicate SFUNC and SPROC values.)

```
Example.         Legal

User A opens the Network with PSD "STAN", SFUNC=0, SPROC=0.
User B opens the Network with PSD "STAN", SFUNC=1, SPROC=0.
User C opens the Network with PSD "STAN", SFUNC=0, SPROC=2.
User D opens the Network with PSD "UCSD", SFUNC=0, SPROC=0.
User E opens the Network with PSD "UCSD", SFUNC=2, SPROC=5.

                 Illegal

User A opens the Network with PSD "GIT ", SFUNC=0, SPROC=0.
User B opens the Network with PSD "GIT ", SFUNC=0, SPROC=3.
User C opens the Network with PSD "GIT ", SFUNC=1, SPROC=0.
User D opens the Network with PSD "GIT ", SFUNC=0, SPROC=0.
User E opens the Network with PSD "STAN", SFUNC=0, SPROC=0.
User F opens the Network with PSD "STAN", SFUNC=2, SPROC=5.
```

Users "A" and "D" both attempt to open the Network with the same PSD, SFUNC, and SPROC values. Only one user can open the Network with a given combination of values at a time. The first user attempting to open the Network with a given combination is granted access. Thereafter, all other users who attempt to open the Network with the same combination will receive an error message until the first user "releases" the combination by closing the Network with NETCLOSE, or uses NETCONTROL to change the combination in effect.

When a program is written to transfer data from one point to another in the Network, you either:

1) code two programs, one to transmit data, and one to receive data (the transmitter is used at one end of your transmission path, the receiver at the other)

2) code one program which both transmits and receives (this program is used at one or both ends).

If you desire to both transmit and receive at the same time, both functions must be performed by the same program. If, however, you are transmitting and receiving batches of records, transmitting a burst, then receiving a burst, two programs (one transmitting, the other receiving) may be used, provided that they do not execute at the same time.

```
Examples.
Key:
   =  Bidirectional transmission
   -  Unidirectional transmission
   >  Information flows to right
   <  Information flows to left
   <> Information flows bidirectionally

      Washington          Network        San Diego

1)    Transmitter >----->=======>-----> Receiver

                           - OR -

      Receiver    <-----<=======<-----< Transmitter

                 But Not Both at the same time!!

2)    Tranceiver  <>===<>=======<>===<> Tranceiver

                           - OR -

      Tranceiver  <>===<>=======>-----> Receiver

                           - OR -

      Tranceiver  <>===<>=======<-----< Transmitter

      But Not More Than One of the Above at One Time!!
```

Please also note that the Network is designed with an option called "non-destructive read" which the Network Administrator can turn on or off. It is recommended that this option be turned on for all systems in the Network.

Non-destructive read allows some overlap time between the time the Network Procedures read a packet, and the time they delete the packet from Network storage, providing a safeguard against data loss due to program or system failure.

This feature can be exploited by the application programmer when handling aborts. How? Well, when the non-destructive read option is enabled, the last packet read via NETREAD is not deleted until either the next NETREAD or NETCLOSE.

Thus, if an application program is reading from the Network and copying to a database and the IMAGE DBPUT call results in an error (such as dataset full), if the application program closes the Network with NETCLOSE mode 2, the last packet read via NETREAD will NOT be lost - instead, it will be the first packet read when the Network is re-opened (with the same SRC, SFUNC, and SPROC parameter values). Again, this "recovery" option (mode 2) on NETCLOSE is only available if non-destructive read is enabled.

When coding calls to the Network Procedures, it is a good idea to follow each call to NETOPEN, NETREAD, NETWRITE, NETCONTROL, NETINFO, NETSTATUS, or NETCLOSE with a check of the first element of the status array returned by the procedures. If the first element (element 0) is non-zero, the procedure call resulted in an error or warning condition. The Network Procedure NETEXPLAIN should then be called to display an explanatory message.

Because the Network Procedures utilize the status array for internal Network information, it is important that the status array be initialized to all zeros prior to calling NETOPEN, and thereafter not be modified in any way by the application program.

Advanced Applications

While the original intent of the Network was to provide a simple, standardized method of transfering transaction data from any point in a multi-system network to any other point in the network, the Network Software is capable of much more.

Utilizing the various parameters available through NETWRITE and NETREAD, applications can be written to handle individual transactions coming from remote points, copy files from point to point, issue MPE commands, and perhaps run programs - all simultaneously. The Network Software can accomodate such an application, provided the application is planned in advance.

To set up the receiving program for such an application, you might do the following:

1) Define a set of codes which will be used to determine the type of packet being received (transaction, file transfer, MPE command, etc.).

2) Define how the codes will be passed (HTYPE or RTYPE parameter perhaps?).

3) Define the heading and data formats to be used for transactions.

4) Define the heading and data formats to be used for file transfers.

5) Define the set of commands which will be available (programmatically executable MPE commands ... and a special routine for :RUN).

6) Define the heading and data formats to be used for commands.

Below is an example

1) Operation Codes:

   1 = Application Transaction - store on application database.
   2 = File Copy - put data in a new file of name and options

specified in HDATA.
3 = MPE Command - perform the requested command.

2) Operation Codes will be passed as HTYPE.

3) Application Transactions are passed in dataset image format.

4) All file copy packets will have the complete file name (including lockword, if any) left justified in the first 36 bytes of HDATA. The first packet of a file transmission will have the text portion of a BUILD command that can be used for constructing the new file in the next 120 bytes.

5) The following commands will be available:
   1) Any MPE command available through the COMMAND intrinsic.
   2) The :RUN command.

   Commands will be passed in COMMAND image format (including the terminating <CR>) in the HDATA parameter.
   RDATA and RTYPE will be ignored.
   If HLEN is zero, the receiving program is to terminate.

As a practical example, consider the following. By using a scheme similar to the above, applications on remote processors can gradually transmit their transactions to the Network as they are generated. The Network transmits the transactions up to the central system, (for example) as the connections permit. Because the transactions are transmitted gradually over an extended period of time rather than as one large batch at the end of the day, the DS lines between the systems are not saturated by large volumes of data monopolizing the DS lines, and lower speed lines may be used. Also, the end of day processing at the central system can begin shortly after the day closes, as there most likely are only a few transactions "in transit", most of the transactions having been transfered during the day, time permitting.

Additionally, using a receiving program similar to that outlined in the previous example, at the end of the day the application could transmit an MPE :STREAM command to launch the end of day processing at the central site automatically after the last transaction is received.

Lessons Learned

While the Network does add to the processing load of a system, its ease-of-use, capabilities, flexibility of configuration, and relatively low maintenance and recovery requirements make it a very useful and productive tool for the applications programmer and user community.

The data transmission efficiency is very dependent upon the size of the data record (or block) passed to the Network. The larger the record (block) transmitted, the less processing the Network is required to perform, and the faster the data is transmitted. Preliminary performance data suggests that transmitting a file of 80 byte records one per packet takes almost twenty times as long as transmitting the same file with twenty 80 byte records in each packet. The point here is that the limiting factor is the number of packets the Network can transmit in a given period of time, not the number of bytes in a packet.

Because the Network Software had to meet the requirement of not loosing any data due to a system failure, certain processing actions had to be taken, all of which limit the ultimate performance of the Network Software.

First, the file labels on all IPC files are updated after every write. This forces the data to disc and protects it, but a tremendous penalty is paid in performance on high speed lines (56kb) where the time spent waiting for the disc I/Os to complete is a performance limiting factor IN TRANSMISSION.

Second, there is the requirement that some sort of logging be utilized to retain at least the last 100 packets transmitted for aiding in recovery after a system failure. This requires MORE disc I/O and incurs additional delays - but it sure does simplify recovery, and reduce the chance of lost data.

Third, transmission statistics for volume analysis. While the statistics the Network keeps on its activity are interesting to say the least, a great deal of time is spent continuously gathering and updating those statistics.

Finally, IPC files don't survive system failures very well unless you force the file label to disc after every write, so many of the above concerns are a moot point. In cases where the forced write has been deliberately disabled (or where the system was running with BLOCKONWRITE = NO), the chances of recovering (intact) an IPC file have always turned out to be slim - very slim - say 10% of the time the file is recoverable.

With the forced writes the IPC files are almost always recovered after a system failure (there was once a case of an IPC file damaged by a system failure that would hang the system any time you tried to access it thereafter), making the Network a safe, reliable, easy-to-use communication subsystem for the HP 3000.

Appendix A

Procedure: NETCLOSE

Purpose:
Used to properly terminate access to the Network.

Parameters:
STATUS ARRAY
Used to return status information to the application program.
MODE

Used to determine how the Network is to be closed - with a non-destructive read (useful for application aborts), or normally.

Procedure: NETCONTROL

Purpose:
Used to change the values of certain user attributes only otherwise changeable by closing and re-opening the Network.

Parameters:

STATUS ARRAY
Used to return status information to the application program.

MODE
Defines which parameters passed to NETOPEN are to be changed. SRC, SFUNC, SPROC, and CIPHER may be changed.

PASS
Password - to verify the identity of the user to the Network.

SYSTEM
Used when passing a replacement SRC value.

SFUNC
Used when passing a replacement SFUNC value.

SPROC
Used when passing a replacement SPROC value.

CIPHER
Used when passing a replacement CIPHER number.

Procedure: NETEXPLAIN

Purpose:
Displays error and status information. Should be called after a Network Procedure returns with the first element of the status array non-zero.

Parameters:

STATUS ARRAY
Used to return status information to the application program.

MODE
Determines whether the error message is to be printed on the $STDLIST device or returned in the buffer supplied by the application program.

BUFLEN
The length of the buffer area used (in bytes). A negative number.

BUFFER
An array to contain the error message returned.

Procedure: NETINFO

Purpose:
Returns information as to the connections, past status of the Network. Information is returned in the same format as NETSTATUS, but information is not as up-to-date.

Parameters:
STATUS ARRAY
Returns status information to the application program.
MODE
Indicates the type and volume of information requested.
SYSTEM
Used when specifying a status request for a specific PSD.
BUFLEN
The length in words (positive) of the buffer returned.
BUFFER
Contains the status information returned.

Procedure: NETOPEN

Purpose:
Opens access to the Network. Must be called before any other Network Procedure.
Parameters:
STATUS ARRAY
Returns status information to the application program.
MODE
Used to tell the Network which "copy" of the Network to use (the software allows for multiple copies of the Network Software so that a production version of the Network can coexist with a test version of the Network).
PASS
User password.
SRC
The source system PSD.
SFUNC
The source function number.
SPROC
The source process number.
CIPHER
The cipher number of the cipher chosen for transmission.

Procedure: NETREAD

Purpose:
Reads packets from the Network.
Parameters:
STATUS ARRAY
Returns the status of the call to the application program.
MODE
Determines the read mode. Reads may be unconditional (if no data is available, NETREAD waits until data becomes available) or timed (NETREAD waits a specific number of seconds before returning a "No data available" warning).
SRC
The source PSD of the packet received.
SFUNC
The source function of the packet received.

SPROC
The source process of the packet received.
FLAGS
The flags set by the sender.
HTYPE
The heading type as defined by the sender.
HLEN
The length of the heading in words.
RTYPE
The record type as defined by the sender.
RLEN
The record length (in words if positive, in bytes if negative).
HDATA
The heading data received.
RDATA
The record data received.

Procedure: NETSTATUS

Purpose:
Returns status information on the lines to the adjacent systems.
Parameters:
STATUS ARRAY
Returns the status of the call to the application program.
MODE
Indicates the type of status request.
SYSTEM
Used to supply the PSD of a system when specifically requesting information on a system.
BUFLEN
The number of words returned to the buffer area.
BUFFER
The area to which the status information is returned.

Procedure: NETWRITE

Purpose:
Writes data to the Network.
Parameters:
STATUS ARRAY
Returns the status of the call to the application program.
MODE
Must always be 1.
DEST
The PSD of the packet destination.
DFUNC
The function of the destination.
DPROC
The process of the destination.
FLAGS
Describes the options used in transmissions (to be continued,

end-of-record, end-of-block, end-of-file, etc.).

HTYPE
The type of the heading transmitted.

HLEN
The length in words of the heading.

RTYPE
The type of record transmitted.

RLEN
The length of the record transmitted (positive if in words, negative if in bytes).

HDATA
The heading data to be transmitted.

RDATA
The record data to be transmitted.

Glossary of Network Terms

In the course of reading this paper,a variety of terms will be used in reference to the Network. Below are some of the most common terms.

Address, Network Address
A multi-field descriptor designating a specific target logical location within the Network. A Network Address consists of 1) node/subnode code (PSD), 2) function, and 3) process.

Extra Data Segment (XDS)
Extra Data Segments refer to areas of memory which through another MPE special capability, DS capability, may be created for storing information, and in some cases, passing information between processes MPE uses extra data segments for storing system tables and file buffers, for example.

Forced Write
A method of forcing the disc copy of a file to be current, including (and most importantly) the label of the file. It is the file label which contains the EOF (end of file) information of a file, and which must be up-to-date if you hope to recover fully from a system failure. A forced write is performed via the MPE FCONTROL intrinsic.

Function (source or destination)
An identifier used to designate the application system the data is coming from or going to. The Function is a positive integer 0 to 32767. The meaning of the different function values is left up to the user community. Both the sender and receiver have a "function" defined. The parameter used in the Network Procedures to specify the sender's function is SFUNC (source function). The parameter used to specify the receiver's function is DFUNC (destination function). Function defines a Network Address in much the same way as street name better defines a geographic address.

IPC
IPC refers to a specific type of MPE file, referred to as an IPC (Inter-Process Communication) file or Message file. IPC files have the unique property of acting as a first-in, first-out (FIFO) queue, making them very useful in inter-process communication.

Local
The LOCAL system is the system which you first log onto and which issues the colon ":" as the MPE prompt.

Network
A collection of CPU's (nodes) running the Network Software package, connected together by DS lines. The Network is divided into nodes, with DS lines running from node to node. Much as the U.S. is a collection of states, the Network is a collection of nodes.

Network Procedures
A set of user-callable procedures used for accessing the Network. The Network Procedures are the applications programmers' only contact with the Network, and are contained in the system SL in two Privileged MPE segments.

Node
A single HP 3000 CPU within the Network. A node is connected to other nodes by DS lines. Each node can be divided into from 1 to 16 subnodes (one subnode, subnode 0 is required for for each node and thus is not user-definable). Only one copy of the Network Software is needed for a Node. Each subnode within the node shares the same Network Software, data base, and files. Node helps to define a Network Address in much the same way state helps to define a geographic address.

Node/Subnode Code
A four character code assigned to a specific node/subnode combination. The Node/Subnode Code is used by the applications programmer when specifying the Network Address of the destination. The Network takes the four character code and translates it to a binary value representing an ordered pair of (node,subnode). The term Node/Subnode code is used where the physical layout of the Network is being described.

Non-destructive Read
One of the unique aspects of IPC files. Normally when a record is read from an IPC file, it is automatically deleted, so that the IPC file acts as a first-in, first-out queue of records. When non-destructive read is enabled (via the FCONTROL intrinsic), the first record of the IPC file is read, but not deleted. Thus, you can obtain and process the record, then post another read (a destructive one) to delete the record, overlapping your processing so that should there be a program or system failure, you

may process the same record twice, but you would not have a record "disappear" into the bit bucket.

Packet

A packet consists of the data the application wishes transferred by the Network, the address of the sender, the address of the destination, a packet ID number, a time stamp, and many other items. The packet is the unit of information which is passed within the Network.

Process (source or destination)

An identifier used in certain Network Procedures to designate the program/process within the applications system specified in "function" that the data is coming from or going to. "Process" is a positive integer 0 to 32767. The meaning of the different process values is left up to the user community. Both the sender and receiver have a "process" defined. The parameter used in the Network Procedures to specify the sender's process is SPROC (source process). The parameter used to specify the receiver's process is DPROC (destination process). Process defines a Network Address in much the same way as house number better defines a geographic address.

Process Handling

Process Handling refers to an MPE special capability. Process handling allows a program (process) to create child processes. These child processes must be activated by the parent process in order to execute. The parent process may also suspend the execution of the child process, or kill the child process. All of these functions are provided to the programmer via several MPE Intrinsics.

PSD

Processing System Designator. A four character code used by the application programmer to tell the Network which logical "system" to use as the source or destination. From the application programmer's perspective, the Network consists of many different PSD's, each effectively on its own CPU. Internally, up to 16 PSD's reside on a single CPU, as the PSD code is translated by the Network Procedures into a (Node,Subnode) ordered pair. In fact, PSD codes and Node/Subnode codes are one and the same - the term PSD is used in logical descriptions of the Network, and the term Node/Subnode code is used in physical descriptions of the Network.

Remote

A REMOTE system is one which you access through the LOCAL system and which issues the pound sign "#" as the MPE prompt. REMOTE systems are further defined by their distance from the LOCAL system.

A REMOTE once removed is a REMOTE directly connected to the LOCAL system. A REMOTE system directly connected to a REMOTE once removed is twice removed, etc.

RIN

RIN is an acronym for Resource Identification Number - a method available through MPE Intrinsics of controlling access to a resource. RINs come in two flavors - Global, which can span job/session boundries, and Local, which can only be shared by processes within a given job/session.

Subnode

One of 16 logical "addresses" within a node. Subnode is a required part of a Network Address. Subnode helps to define a Network Address in much the same way as city or county better define a geographic address.

3049. KSAM Survival Techniques

Dennis Scheil
Base 8 Systems, Inc.
21 Grist Mill Road
Belle Mead, N.J. 08502
(201) 874 - 8887

The focus of this paper is on KSAM files, how they work and why, sometimes they do not work. This paper will NOT try to tell you that KSAM flat-out does not work. It will not tell you to throw KSAM away and use IMAGE in all circumstances ... including generic key processing! On the other hand, this paper will not sell KSAM as the greatest thing since the integrated circuit, nor will it attempt to convince you to give IMAGE the heave-ho and roll in the KSAM files. As usual, the truth about KSAM is somewhere in the middle, between the extremes. This paper will give you some KSAM survival techniques: first, the information you need to understand KSAM and the File System; then ways of making KSAM work for you; next some comparisons between KSAM and IMAGE and finally some 'tricks of the trade' to make life with KSAM much easier. As a side benefit, should you choose KSAM someday, this paper may help you justify your selection to your peers and management when they look at you in askance and say 'WHAT? You used KSAM???'.

First, let's take a look at the structure of a KSAM file. As is well known, a KSAM file actually consists of TWO separate MPE files. The first, the DATA file, is simply a standard sequential file; it doesn't even have a special file code. Records are stored in the data file 'chronologically'; that is, in the sequence they were written. The only 'nonstandard' feature of this file is a 'user file label' containing the name of the second file, the KEY file. The key file, as the name implies, holds the key values for records in the data file and the record addresses of these data records. It also contains internal pointers to locate the next and previous key values; these pointers are used to locate records by key and also for 'sequential' access to the data. Note that sequential access to a KSAM file is actually a sequential reading of the KEY file, not the data file. The key entries are arranged in a structure known as a 'B-tree'; the B-tree consists of levels of key entries called 'key blocks'. These key blocks will be discussed later in the paper.

The key file also contains control information such as the data file name, number of file accesses, create and last access dates, definition of each key and the KSAM end of file pointers for both files. This information may be displayed using KSAMUTIL's VERIFY function against a given KSAM file. The key definitions or 'descriptors' specify the data type for the key, key length, start position in the data record and a pointer to the 'root' key block in the B-tree.

The key file has a file code of 1080 (mnemonic KSAMK) and is always a binary file with 128 word, fixed length records. The file limit of the key file is established by the File System when the file is created and is not under user control. The MPE end-of-file pointer is set at the file limit, as it is with an IMAGE dataset; thus, the entire file is allocated at once. As with IMAGE, KSAM maintains its own internal EOF for the key file.

Clearly, there are a number of linkages within the key file, and one for each active data record between the key and data files. When a program accesses a KSAM file, it is imperative that both files be kept current. The File System handles the link and pointer maintenance in a special KSAM extra data segment, or EDS. One KSAM EDS exists for EACH open KSAM file in the system. The data segment contains a control and key descriptor block, a working-storage area, the current data block buffer and from one to twenty key block buffers. The extra data segment may be as large as 32K words, but from 6 to 8K is typical for a single-key KSAM file. When the EDS is created, 12K of memory is allo- cated; if less space is required, the REAL memory allocation is reduced while the VIRTUAL remains at 12K. If MORE than 12K are required, the data segment is 'released' (purged) and a new, larger one created.

The control block portion of the KSAM EDS contains a copy of the control and key information from the key file and is updated each time the file is accessed. The key file itself is only updated when the KSAM file is unlocked or closed.

The data and key buffer blocks are refreshed or written as required. The need for a new data buffer, for example, does not force a read of the key file also. The buffer area is cleared whenever the file is locked, forcing reads from disc, and changed data are posted to disc whenever the file is unlocked or closed.

Thus, the important components of a KSAM file are:

1. The data file;
2. The key file - control and key descriptor area;
3. The key file - key blocks; and
4. The extra data segment created whenever a KSAM file is opened.

Let us now turn to the File System to explore the wonders of shared file access, locking and buffer allocation.

The MPE File System permits three different types of shared access to sequential and KSAM files: SHR [;NOMULTI], SHR;MULTI and SHR;GMULTI. The 'SHR' keyword simply permits more than one process to access a file concurrently. The GMULTI, MULTI and default NOMULTI parameters specify the type of buffer sharing to be in effect for a given accessor. Under GMULTI access, buffers and pointers are shared between processes. MULTI access is similar to GMULTI except that the sharing of buffers is permitted only within one job/session, mainly useful in 'process handling' environments. NOMULTI access, the default if neither MULTI nor GMULTI are specified, means that each process maintains its own set of buffers for each open file. If four processes share a file, under GMULTI (or MULTI) access, all would share the same buffer and control block. If NOMULTI access were used, each process would have its own buffer and control block. If three processes specified GMULTI and the fourth opted for NOMULTI, two buffers would exist for the file; the first three (GMULTI) processes would share one and the NOMULTI process would own the second buffer exclusively. Which access method is better, then? Well, it depends on what the process wishes to do with a file.

Any time a file is shared and ANY process reads that file sequentially, DO NOT USE GMULTI ACCESS! Under GMULTI, two programs reading the same file simul- taneously will get exactly HALF the records each! Assuming both are reading at the same rate, one program will get the 'odd' records and the other the 'even' ones. The average system user will not understand what is going on. The auditors will not be amused. Because GMULTI accessors share the same control block, the 'current record pointer' in the file is advanced by BOTH processes, so after program A reads record 1, it advances the pointer and program B gets record 2. Throw program C into the file doing RANDOM reads and one suddenly realizes that GMULTI is not going to work here. So, to save time, effort and lots of perspiration, use NOMULTI access when reading any MPE or KSAM file sequentially.

The file system permits processes WRITING to a file to share it without any locking whatsoever. Obviously, all processes must open the file for APPEND access in this case. Here, GMULTI access is essential, since all processes MUST know where the current record pointer (EOF!) is and should share a common buffer to prevent buffer collisions. Unfortunately for the KSAM user, this 'unlocked append' access is not available as KSAM forces either exclusive access or locking before updating a file. The locking requirement, which likewise assures current pointers and no buffer collisions, virtually eliminates this need for GMULTI access to KSAM files. There is a way to make GMULTI access work with simultaneous sequential and random processing, however and we will geet to it after discussing one of the least understood aspects of the File System, namely LOCKING.

In order to lock a shared file, it must have been opened with 'dynamic locking enabled', either by setting bit 10 of the 'aoptions' parameter to FOPEN or by specifying the ';LOCK' parameter in a file equation for that file. The COBOLII compiler generates the dynamic locking access option for file having an EXCLUSIVE statement. Once a file has been opened with locking enabled, ALL other accessors must do the same. Conversely, if the file has been opened NOLOCK, then subsequent accessors may not request dynamic locking. Violators receive FSERR 48, 'INVALID OPERATION DUE TO MULTIPLE FILE ACCESS', do not pass GO and do not collect $200. Ahhh. Now the system analyst (and DP manager) can sleep at nights. Just specify ;LOCK on all files and SHAZZAM! No more worries. Right? WRONG! The battle has just begun.

Once a shared file is open, there is NOTHING in the File System to prevent a 'reader' from accessing the file while another accessor has locked it! The FLOCK intrinsic, the EXCLUSIVE statement, the CKLOCK and BKLOCK KSAM procedures, even the old COBOLLOCK, NONE of these prevent other users from reading a 'locked' file. The only thing that the FLOCK intrinsic will do is prevent ANOTHER FLOCK from succeeding! This does NOT mean that a two processes may update the file simultaneously, since the dynamic locking option forces a lock before an update. It DOES mean that there is no way of preventing a 'reader' from accessing the very same record that another process is updating UNLESS ALL READERS LOCK BEFORE READING! Why? The answer lies in the way the File System 'locks' a file.

When the FLOCK intrinsic is called, either implicitly (EXCLUSIVE, CKLOCK) or explicitly, the File System attempts to lock something called a GLOBAL RIN (RIN stands for Resource Identification Number). A global RIN is assigned to any file opened wih dynamic locking enabled. A process trying to lock the file is actually trying to lock the global RIN assigned to that file, not the file itself. If two processes attempt to lock the same RIN, the second process will be impeded until the first releases the lock, unless CONDITIONAL locking was specified. In that case, the second lock attempt will fail. Once the file (RIN) has been locked, other processes wishing also to lock the file will have to wait until the lock is released by a call to FUNLOCK. But what happens if a process does not call FLOCK and just tries to read the file while it is locked? NOTHING! The non-conforming program simply breezes past the processes waiting patiently for the RIN, barges past the locking process and waltzes away with an I/O buffer full of records, records which may or may not be valid. The locked RIN did NOTHING to impede the non-locking process.

Should an application need to update TWO files simultaneously and wish to lock both of them to ensure that they remain in synch, the usual result is FSERR 64 USER LACKS MULTI-RIN CAPABILITY. The File System does not want to lock two files at once and will prevent this from happening unless the locking program is PREPped with MR (Multi RIN) capability. 'Fine', you say. 'We'll just let all of our programmers have MR capability and let them lock as they please. Right?' WRONG, unless you are training your operations staff on shutdown and startup procedures. Multi RIN capability is considered by HP to be the second most dangerous special capability (right behind that old devil, Privileged Mode) not because is breeches security or causes system failures, but because it can create a situation known as 'deadlock', from which a system shutdown is the only recovery. Actually, consider it a CONTROLLED SYSTEM FAILURE, controlled because it gives system management time to tell users to log off but a system failure nonetheless because it forces you to halt the system. The classic deadlock scenario: Process A has locked File 1 but is prevented from locking File 2 because Process B has locked File 2 but is prevented from locking File 1 because Process A ..... (repeat ad nauseum).

The first rule of MR capability is: DON'T! unless you absolutely, positively must. The second rule: If you've gotten this far, read rule 1 again. The third rule: Well, if you insist, just remember that ALL PROCESSES MUST LOCK FILES IN E X A C T L Y THE SAME ORDER AND RELEASE THE LOCKS IN THE SAME ORDER. Usually, installations follow the ASCII collating sequence, locking file A before B before X. And then one day you hire a new programmer and forget to tell her the locking sequence and her old employer used REVERSE collating sequence locking . . . Murphy LOVES 'MR' capability, but the use of large numbers of KSAM or sequential files often need MR to keep the files synchronized. Since IMAGE-based systems usually combine a number of files into one database, they do not require MR as often; if you need to keep three datasets in synch, just DBLOCK the whole database (mode 1 or 2). Of course, if you need to lock two databases, or a database and a KSAM file, MR may be the only way to go (although IMAGE does not use RINs, it respects the 'one lock at a time' rule). But, wherever possible, apply The First Rule of MR and DON'T! To be successful, Multi RIN capability must be very well planned, executed and controlled.

Let's sum up the main points to consider in sharing MPE files between processes:

1. Sharing a file is enabled using the SHR keyword in a file equation or specifying the 'share' access option to the FOPEN intrinsic;
2. A file may be accessed with SHARED buffers and pointers if it is opened with GMULTI access. This is NOT advised for sequential readers;
3. Use of NOMULTI access means that each process has its own buffers and control block for a shared file. This access mode may be dangerous if processes are updating the file without locking and very questionable if updates are occurring while other programs are reading the file without locking it;
4. Locking is controlled not by physical locks to a file but rather by using RINs (Resource Identification Numbers). When a process 'locks' a file, it obtains the RIN for that file. Subsequent attempts to lock the file will fail or be impeded until the lock is released. File readers who do not lock before reading are NOT impeded; thus the lock mechanism is not even close to fool-proof; thus
5. To ensure data and report integrity, ALL processes must lock a shared file before performing ANY operation on that file unless GMULTI access is used. In this case, extreme care must be taken to ensure data integrity for the 'reader' processes and locking is required for sequential reads anyway; and
6. Locking more than one file at a time requires MR (Multi RIN) capability which can be dangerous. Uncontrolled multi file locking WILL lead to process deadlocks which require a system shutdown to clear.

It is readily apparent that the File System does not simplify multi-user access to data files. Add in the complexity of KSAM and one begins to understand the full import of HP's warning on sharing files:

'SHARING A FILE BETWEEN TWO OR MORE PROCESSES
MAY BE HAZARDOUS' (File System manual, p 5-16)

Now, let's take a look at one of the important places where KSAM meets the File System and examine the implications of GMULTI and NOMULTI access. Remem- ber that KSAM files require an extra data segment in addition to the normal set of file buffers and control blocks for the key and data files. When a SECOND process opens a shared KSAM file and GMULTI access is specified, both users share the buffers, control blocks AND the KSAM data segment. If NOMULTI access is requested, a new set of KSAM overhead must be created. Now we have TWO extra data segments and two sets of buffers and control blocks. If TEN users share the same file NOMULTI, then TEN data segments are used. If each of the ten processes shares TWO KSAM files, TWENTY DATA SEGMENTS ARE REQUIRED! No wonder that HP states: 'KSAM files can use a lot of memory.' (KSAM manual, p. B-17) The 20 data segments would, if all were present in memory at the same time, use from 100K to 240K words of memory, UP TO ONE-EIGHTH OF THE MAXIMUM AVAILABLE MEMORY ON A SERIES III MACHINE, and ten users are not an unrealistic number for a Series III. Of course, all of these segments would probably not be in memory at one time, which implies a much greater workload for the Memory Manager as segments are swapped out and rolled back in from disc. A Series III, with its single I/O channel is now not just I/O bound, it is I/O swamped! Even a 44 can get shaken up by a big KSAM-imposed workload.

The solution may well be GMULTI access; since there would be only one data segment per file, much less memory would be used and that segment, shared between several processes, would probably never be swapped out. Unfortunately GMULTI doesn't work with sequential access unless the file is locked for the entire sequential read, or a special LOCK-START-READ-UNLOCK technique is used. The former, a 'long lock', would cut dramatically into other users' response times and is not really viable. The LOCK-START-READ-UNLOCK will work even with simultaneous sequential and random processing. It entails some extra programming, but may be worthwhile in the long run, especially if main memory is a problem and GMULTI access is the solution.

Instead of using a simple READ (or COBOL 'READ ... NEXT) to retrieve records sequentially, a sequence of commands is issued. A LOCK (EXCLUSIVE, CKLOCK or FLOCK) is issued, followed by a START. The START uses the key of the last record read and a relative operator of 'greater-than'. This positions the KSAM logical record pointer at the next sequential record. The READ (or a series of READ's) is issued next, followed by an UNLOCK to relenquish the file. If GMULTI access is to be used and a given KSAM file may not be locked for more than a bufferful of records, this is the only way to access the file sequentially. If this method is to be used, it is ESSENTIAL that all reader processes lock the file before reading, as a pointer-moving call between the LOCK and the READ would cause all kinds of grief. Two additional points to consider are that after the file open, the key field should be primed with low-values and that the EOF indication will come from the START, not the READ.

A third solution, and probably the best one, is to give sequential processors NOMULTI access to a file and use GMULTI for random and on-line accessors. The MULTI/NOMULTI option is mix and match, allowing both types of access to the same file at the same time, unlike the selection of dynamic locking, where all accessors must agree. GMULTI file accessors will share a common buffer amongst themselves and NOMULTI accessors will have individual buffers.

Note that the CKOPENSHR opens a KSAM file for 'SHR;LOCK [;NOMULTI]'.

Another, equally important consideration for sharing KSAM files is the use and effect of locking. With sequential files, FLOCK doesn't just lock the file, it also clears the file buffers to force the next read to initiate a transfer from disc. FUNLOCK flushes the file buffers, ensuring that the disc has been updated before the next process accesses the file. KSAM files require one additional step: the refreshing of the extra data segment. The following quotes excerpts from the KSAM manual illustrate this additional step (and make another strong point for file locking for all accessors):

> 'When FLOCK is executed, it clears all the buffers and transfers the latest control information from the KSAM file to the buffers. This ensures that any subsequent read of the file retrieves the latest information from the disc rather than from the buffers.' (KSAM manual, p 4-39)
>
> 'When FUNLOCK is executed, all output written while the file was locked is transferred to the file [from the buffers] so that other users have the most recent data.' (KSAM manual, p 4-91)

'Because the current pointer position is not in a "common block", when several programs open the same file, each can alter the key file structure by adding or deleting records so that pointers set by other programs may point to the wrong record without those other programs being aware of it.
'To make sure that the latest pointer position is stored with the file rather than in the separate extra data segments, programs that share the same KSAM file must use a locking scheme ... [E]ach program should lock a KSAM file before executing any procedure that positions a pointer ... and not unlock the file until all procedures that depend on this pointer position have completed execution.' (KSAM manual, p B-21)

Note that there are two kinds of pointers in an open KSAM file, the internal key file pointers and the 'current record' pointer. BOTH of these are critical to the integrity of any application accessing the file in a shared environment. The meaning of the above excerpts is clear: a locking strategy MUST be in place for ALL KSAM file accessors. If you are STILL not convinced of the need to lock, even for read access, let the following sentence from the KSAM manual and practical example do the job:

'When a key file is searched for a particular record, the root block and lower level blocks, AS NEEDED are moved to the key block buffers in the [KSAM] extra data segment. (caps mine)
(KSAM manual, p B-17)

The data and key buffers are refreshed ONLY AS REQUIRED (and FLOCK will force that requirement). If the file is not locked, the disc is not accessed until either the key or data buffers do not contain the required record. Then, ONLY THE BUFFER NEEDING THE DATA will be updated. To illustrate, program A reads a KSAM file sequentially, displaying each record on a termainal and pausing until the user indicates she has read the data. Then, the next record is read and displayed. Program B also reads the file sequentially but deletes them. Program A does not lock the file, program B locks before each read and unlocks after the delete. In this example, program A is run and the user is looking the first record in the file. Now, someone runs program B and deletes all of the records in the file! What will happen to A when the user asks to see the

next record? OOOOOOPS! The user GOT the next record, even though program B had deleted it! In fact, UNTIL THE KEY BUFFERS ARE EXHAUSTED, PROGRAM A WILL CONTINUE TO READ RECORDS AND DISPLAY THEM AS IF THEY WERE ACTIVE! If the key buffers hold more entries than the data buffer (a likely occurrence) program A will charge blithely on, even refreshing its data buffer with deleted records. There will even be a delete flag in the first two bytes of each record in subsequent data blocks! This scenario assumes NOMULTI access; had GMULTI been in effect, program A would have found itself at end-of-file (more or less correctly so), but program B would have skipped the first record in the file. Had A requested a record in the middle of B's deleting, it would have gotten one from somewhere in the middle of the file and B would have missed that one also! If both programs had used locking before reading (with either GMULTI or NOMULTI) program A would hav correctly reached EOF on the second read and B would have deleted all of the records. If A had requested another record before B deleted all of them, A would have gotten the first un-deleted record, but B would have been able to delete it afterwards.

This leads us again to the First Rule of KSAM:

> ALL ACCESSORS OF A SHARED KSAM FILE SHOULD LOCK THE FILE PRIOR TO ANY I/O OPERATION, INCLUDING READS!

There is, of course, one exception to this rule; if no updates are being performed and all users are accessing the file NOMULTI, locking is not required.

There are two serious complications arising from KSAM's stringent locking requirements. First, if a system is I/O bound before locking, it will be even more so afterwards. Since FLOCK forces the next read to come from the disc and FUNLOCK posts any modified buffers back to the disc, the blockfactor of the data AND key files effectively becomes '1'. KSAM will exacerbate the situation by requiring at least three I/O's per read or write, two to the key file and one to the data file. The reason that two I/O's will be required to update the key file is that the KSAM control information must be updated also. Sequential file read processes may optimize a bit by reading more than one record per file lock, but random readers and file update programs can not be helped by this technique. A secure KSAM system will probably be I/O bound.

The second problem concerns the co-ordination of locking when more than one file must be locked. Since KSAM requires the locking of INPUT files, the need for Multi RIN capability becomes acute, but the problems make its use very unattractive. One solution may be to combine several KSAM files into one; this approach would possibly reduce memory requirements also, but it has the definite drawback of creating a bottleneck for system users. While throughput suffers less with a bottleneck than with a deadlock, neither option seems very palatable. Obviously, designing or implementing a KSAM-based system requires special care and planning, most emphatically so whenever MR capability will be required to lock more than one file at a time.

At this point, recognizing the flaws of KSAM, one might ask if it is worth using it at all. IMAGE must be better in all cases, right?

First, it is not entirely KSAM's fault. The design of the MPE File System is not really satisfactory for the sharing of data between processes, especially in a dynamic on-line environment. GMULTI access results in pointer ping-pong but NOMULTI means big trouble if someone is updating at the same time. Still the biggest problems occur while different types of access are occurring. If a KSAM file is accessed entirely for update or solely for sequential reads, no problems occur. Therefore, it is wise to consider some advantages of KSAM over IMAGE and use it under 'controlled' circumstances.

Typical uses for KSAM are extract files, tables and 'batch files' used to collect data on-line for subsequent batch database update. In all cases, processing is well defined and well suited to KSAM. KSAM is much faster than IMAGE for many types of processing. Slow IMAGE batch programs may be speeded up significantly by using a utility program such as Robelle's SUPRTOOL to extract data from an IMAGE database into a KSAM file and then running against the extract instead. While it seems that much more overhead is involved, run times may actually be cut in half and this includes the time required to perform the extract! KSAM is equally well suited for tables systems, especially those built around generic key processing.

Note that none of these functions involve the 'core' data of a system. KSAM is NOT a wise choice, especially when several files are required and updates must be performed against more than one at a time. There are many types of 'peripheral' processing, however, for which KSAM is the logical choice.

Of course, none of this is very comforting to the MIS manager stuck with a cranky KSAM system. There are steps, however, which may be taken to ensure better throughput (and better OUTPUT!) without trashing the system and rewriting it in IMAGE.

First: Make sure that ALL on-line update programs are using GMULTI access to update the file. Then check the batch programs to make sure that they are running NOMULTI. Make absolutely sure that any batch job which may run against KSAM files when there is on-line activity specifies SHR;LOCK in a file equation. Otherwise, you may not get very far, either with the batch job or the on-line system.

Second: Check your locking stategies. If locks are being applied for update only, you may need to restrict batch processing to certain hours of the day. You may want to try rewriting the I/O routines to make sure that all locks are being applied correctly. Isolate those programs which use MR capability and make sure that all locks are applied in the same order. Make sure that all of these locks are necessary and get rid of those that aren't. If any batch programs use MR and run during on-line access hours, change the jobstreams to create extract files instead and get rid of the multiple locks.

Third: Set up two jobstreams, one called CRASH and the other called CLEAN. CRASH will use KSAMUTIL's KEYINFO function to check every KSAM file in the system for structural damage after a system failure. Just specify 'KEYINFO filename', the RECOVER parameter is not necessary. Then tell Operations to stream CRASH after EVERY system failure and not to let anyone sign on until CRASH has finished. CLEAN will use FCOPY to copy records to a sequential file and then back to the KSAM file. This operation removes deleted records AND puts the records back in key sequence. Even if you are reusing deleted record space, the CLEAN step is necessary to put the records back in order.

Fourth: Keep a close watch on file capacities, ESPECIALLY if you are not reusing record space. Don't make the files too large, but be generous.

KSAM may not be perfect, in fact it is very far from that, but there is no reason to reject it in situations where it will work just fine. Likewise, unless the application is in very bad shape, a few changes to an existing KSAM system may make life with KSAM a bit more livable.

3050. TURBO PASCAL AND AGIOS ON THE HP150

Steve Porter, CDP
D P Systems
Box 34429
Memphis, Tennessee 28134

WHAT IS TURBO PASCAL

Pascal is the general-purpose high level programming language originally designed by Niklaus Wirth of the Technical University of Zurich, Switzerland. He named the language in honor of Blaise Pascal, the French philosopher and mathematician.

Turbo Pascal is designed to meet the requirements of all categories of users. It follows the definition of Standard Pascal as defined by K. Jensen and N. Wirth in the "Pascal User Manual and Report" quite closely. In addition to the standard, a number of extensions are provided. Among these are:

* Absolute address variables * Bit/byte manipulation Direct access to CPU memory and data ports Dynamic strings Free ordering of sections within declaration part * Full support of operating system facilities In-line machine code generation * Include files Logical operations on integers Program Chaining with common variables Random access data files Structured constants Type conversion functions

These extensions are what makes it relatively easy to access the AGIOS subsystem.

WHAT IS AGIOS?

AGIOS stands for Alpha-numeric Graphics Input Output System. It is a layer of software between BIOS (Basic Input/Output System and the MS-DOS operating system. From MS-DOS AGIOS appears as a device driver. It is designed to give you, the programmer, total high-speed control over the HP150 without resorting to direct hardware-specific calls and addresses. HP has made a commitment to maintain compatibility in future systems in the series 150 line. The 110 does not appear to be included in this commitment.

WHY SHOULD I USE AGIOS?

There are two general reasons why you might want to use AGIOS. The first is speed, and the second is that there are some things that there is just no other way to accomplish. According to HP specs the standard console output rate is approximatly 700 characters per second. AGIOS, on the other hand, allows console output as fast as

4000 characters per second. For approximately 1/3 of the functions, AGIOS is the only way you can do that function.

HOW TO ACCESS AGIOS FROM TURBO PASCAL

Because AGIOS is designed as a device driver, the AGIOS functions are all accessed through MS-DOS calls. Specifically, the 'I/O Control Write' on the console device. This use of standard MS-DOS calling conventions allows you to access AGIOS from Turbo Pascal by using the standard procedure "MsDos". This procedure, which is one of the extentions provided by Turbo Pascal:

1) allows you to set up a record with all of the data necessary for an MS-DOS call, and 2) executes an interrupt 21 (function-request) call to MS-DOS.

The format of the record to be used by the procedure is:

```
REGISTERS = RECORD AX:        INTEGER; BX:          INTEGER; CX:
   INTEGER; DX:               INTEGER; BP:          INTEGER; SI:
   INTEGER; DI:               INTEGER; DS:          INTEGER; ES:
   INTEGER; FLAGS:    INTEGER; END;
```

The values for the registers are as follows:

```
AX := $4403;   {The 44 tells DOS that this is a 'I/O Control Write'
                The 03 tells the 'I/O Control Write' that it is to
                write CX number of bytes from the buffer pointed to
                by the DS:DX pair to the device control channel}

BX := 1;       {The Handle of the device, in this case the console}

CX := length   {The length of the buffer - see AX}

DX := offset   {The offset of the buffer within the segment}

DS := segment  {The segment of the buffer}
```

If there is a MS-DOS problem the carry-flag will be set and AX will have a error number in it. The errors are:

1 = invalid function, 5 = access denied, 6 = invalid handle, and 13 = invalid data.

If the carry flag is not set and AX = 0 then an AGIOS error has occurred. Normally the AX will return with the number of bytes transferred. This covers the mechanics of the MS-DOS calls. Next we will look at the nuts & bolts of how the buffers are laid out for each function.

AGIOS - THE NUTS AND BOLTS

All of the AGIOS functions are performed through the same MS-DOS function. The only difference between the calls is the buffer that is written to the device driver. Even though the format of the buffers can vary widely in both layout and size, they all have some parts in common. The general layout of the buffer is:

```
FUNCTION_BUFFER  :  RECORD  FUNCTION_SUBTYPE:                 BYTE;
        FUNCTION_TYPE:              BYTE; the remainder of the buffer is
        dependent on the function.
```

There are two FUNCTION_TYPE's. They are "0" for the Alpha/Numeric functions and "1" for the Graphics functions. There are over 100 FUNCTION_SUBTYPE's distributed between the two function types. The following is a list of all available functions:

----------- ALPHA/NUMERIC -----------

Video intrinsics
- * DEFINE AREA
- * WRITE AREA
- * CLEAR AREA
- * ENHANCE AREA
- * READ AREA
- * SHIFT AREA
- * WRITE LINE

Control functions
- EXECUTE TWO-CHARACTER SEQUENCE
- POSITION CURSOR
- DEFINE ENHANCEMENTS
- CURSOR SENSE ABSOLUTE
- CURSOR SENSE RELATIVE
- * SET CURSOR TYPE
- * READ CURSOR TYPE
- * READ TERMINAL CONFIGURATION

Keyboard intercept
- * DEFINE KEY CHARACTERISTICS
- * GET KEY STATUS
- * PUT KEY
- * KEYCODE ON/OFF
- * KEYCODE STATUS
- * READ KEYPAD STATUS

Application softkeys
- * UPDATE SOFTKEY LABEL
- * READ SOFTKEY LABEL
- * DISPLAY SOFTKEY LABELS

Touch screen functions
- DEFINE TOUCH FIELD
- DEFINE SOFTKEY FIELD
- DELETE TOUCH FIELD
- SET TOUCH SENSING MODES

* These functions do not have a equivalent escape sequence, therefore are only available through AGIOS.

----------- GRAPHICS -----------

Display control
- CLEAR GRAPHICS MEMORY
- SET GRAPHICS MEMORY
- TURN ON/OFF GRAPHICS DISPLAY
- TURN ON/OFF ALPHANUMERIC DISPLAY
- TURN ON/OFF GRAPHICS CURSOR

Graphics plotting
- LIFT PEN
- VECTOR MOVE ABSOLUTE
- VECTOR MOVE INCREMENTAL
- VECTOR MOVE RELOCATABLE
- LOWER PEN

TURN ON/OFF RUBBER BAND LINE
MOVE GRAPHICS CURSOR ABSOLUTE
MOVE GRAPHICS CURSOR RELATIVE
TURN ON/OFF ALPHANUMERIC CURSOR

TURN ON/OFF GRAPHICS TEXT MODE

Vector drawing mode
- SELECT DRAWING MODE
- SELECT LINE TYPE
- DEFINE LINE PATTERN/SCALE
- DEFINE AREA FILL PATTERN
- FILL RECTANGULAR AREA ABSOLUTE
- FILL RECTANGULAR AREA RELOCATABLE
- SELECT POLYGONAL FILL PATTERN
- SELECT BOUNDARY PEN
- NO POLYGON BOUNDARY
- SET RELOCATABLE ORIGIN
- SET RELOCATABLE ORIGIN TO PEN POSITION
- SET RELOCATABLE ORIGIN TO CURSOR POSITION
- SET GRAPHICS TEXT SIZE
- SET GRAPHICS TEXT ORIENTATION

- TURN ON/OFF TEXT SLANT
- SET GRAPHICS TEXT ORIGIN
- GRAPHICS TEXT LABEL
- * DEFINE USER CHARACTER SET
- * SELECT DEFAULT CHARACTER SET
- * OUTPUT SINGLE CHARACTER
- SET GRAPHICS DEFAULT
- SET PICTURE DEFINITION DEFAULTS
- GRAPHICS HARD RESET
- VECTOR DRAW ABSOLUTE
- VECTOR DRAW INCREMENTAL
- VECTOR DRAW RELOCATABLE
- SET PEN POSITION TO CURSOR POSTION
- POINT PLOT
- SET RELOCATABLE ORIGIN TO PEN POSITION
- START POLYGONAL AREA FILL
- TERMINATE POLYGONAL AREA FILL
- * POLYGON MOVE ABSOLUTE
- * POLYGON MOVE INCREMENTAL
- * POLYGON MOVE RELOCATABLE
- * POLYGON DRAW ABSOLUTE
- * POLYGON DRAW INCREMENTAL
- * POLYGON DRAW RELOCATABLE
- LIFT BOUNDARY PEN
- LOWER BOUNDARY PEN

Graphics status
- READ DEVICE ID
- READ PEN POSITION
- READ CURSOR POSITION
- READ CURSOR POSITION WAIT FOR KEY
- READ DISPLAY SIZE
- READ GRAPHICS CAPABILITY
- READ GRAPHICS TEXT STATUS
- READ ZOOM STATUS
- READ RELOCATABLE ORIGIN
- READ RESET STATUS
- READ AREA SHADING
- READ DYNAMICS
- * READ EXTENDED SCREEN DIMENSIONS

* These functions do not have a equivalent escape sequence, therefore are only available through AGIOS.

The folowing program shows how some of the AGIOS functions may be used. A copy of this program and other routines to access the AGIOS functions is in the INTEREX contributed library on CompuServe. The program will draw a 'calendar' using the Line Drawing Character Set. The program will also use the ROMAN BOLD Character Set that can only be accessed through the AGIOS functions. The program demonstrates the speed of the AGIOS functions by filling the screen, including enhancements, in less than 1/2 of a second.

```
PROGRAM TESTINTR;

{************************************************************************
* This program tests some of the video intrinsics                       *
************************************************************************}
```

```
{***********************************************************************
*                  Copyright 1984 by Steve Porter, CDP                 *
*                                                                      *
* Permission is granted for anyone to use these programs for private or*
* commercial use provided that the following notice is included:       *
*                                                                      *
*   'Portions of this program are copyrighted by Steve Porter 1985'    *
*                                                                      *
***********************************************************************}


{***********************************************************************
*  Some general TYPEs needed by all routines                           *
***********************************************************************}
TYPE

  REGISTERS = RECORD
    AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS: INTEGER;
  END;

******* IO_CT ******


{***********************************************************************
* This function does an I/O Control operation and dumps the flags if   *
*   the operation fails                                                *
***********************************************************************}
PROCEDURE IO_CTL(VAR BUFF; LEN: INTEGER);

    VAR
        REG_PACK : REGISTERS;

    BEGIN
        WITH REG_PACK DO BEGIN
            AX := $4403;  {I/O CONTROL OPERATION}
            BX := 1;      {CONSOLE HANDLE (ALWAYS 1)}
            CX := LEN;    {BUFFER LENGTH }
            DX := OFS(BUFF);
            DS := SEG(BUFF);
        END;
        MSDOS( REG_PACK );
        IF REG_PACK.FLAGS AND 1 <> 0 THEN BEGIN
            WRITELN( '*********************************************' );
            WRITELN( 'MSDOS ERROR ' );
            WRITELN( 'FLAGS: ', REG_PACK.FLAGS, ' AX: ', REG_PACK.AX );
            CASE REG_PACK.AX OF
                6: BEGIN
                       WRITELN( 'Invalid Handle ' );
                   END;
                1: BEGIN
                       WRITELN( 'Invalid Function ' );
                   END;
                13: BEGIN
                       WRITELN( 'Invalid Data ' );
                END;
             5: BEGIN
                    WRITELN( 'Access Denied ');
                END;
         END;
         WRITELN( '************************************************' );
      END
      ELSE BEGIN
         IF REG_PACK.AX <> 0 THEN BEGIN
             WRITELN(
              '************************************************' );
             WRITELN( 'AGIOS ERROR ' );
             WRITELN( 'FLAGS: ', REG_PACK.FLAGS, ' AX: ',
              REG_PACK.AX );
             WRITELN(
              '************************************************' );
         END;
      END;
   END;

{*******************************************************************
* This routine and its variables are used to control where the    *
* Graphics intrinsic output is to go.  The Options are as follows  *
*  0 - display to screen  ( default )                              *
*  1 - display to both screen and plotter                          *
*  2 - display to plotter only                                     *
*******************************************************************}

VAR
    PLT_DEVICE:    TEXT;
    PLT_MODE:      INTEGER;
    PLT_ASSIGNED:  INTEGER;
    PLT_SCALE:     INTEGER;

PROCEDURE PLOT_CONTROL( PLOT_TYPE: INTEGER );
    BEGIN
        IF PLT_ASSIGNED = 1 THEN BEGIN
            CLOSE( PLT_DEVICE );
        END;
        IF PLOT_TYPE <> 0 THEN BEGIN
            IF PLT_ASSIGNED <> 1 THEN BEGIN
                ASSIGN( PLT_DEVICE, 'PLT:' );
                PLT_ASSIGNED := 1;
            END;
            REWRITE( PLT_DEVICE );
            WRITE( PLT_DEVICE, 'IN;' );
            PLT_SCALE := 20;
        END;
        PLT_MODE := PLOT_TYPE;
    END;

{***********************************************************************
*  Some general TYPEs used by the VIDEO INTRINSICS                     *
***********************************************************************}
```

```
TYPE
  AREA_POINT = REA_DEF;  { This is the TYPE returned to show the
                            previous area }
  AREA_DEF = RECORD
    LR_ROW,
    LR_COL,
    UL_ROW,
    UL_COL: BYTE;
  END;

{***********************************************************************
* This function specifies the area to be operated upon by subsequent    *
* area update operations.                                               *
*   INLR_ROW & INLR_COL  Defines the lower right corner of the area     *
*   INUL_ROW & INUL_COL  Defines the upper left corner of the area      *
*                                                                       *
* This function returns a buffer of type AREA_POINT that has the prev-  *
* ious coordinates in it.                                               *
***********************************************************************}

FUNCTION DEFINE_AREA( INLR_ROW, INLR_COL, INUL_ROW, INUL_COL  : BYTE ):
 AREA_POINT;

  VAR
    OLD_AREA           : AREA_POINT;

    DEFINE_AREA_BUFFER : RECORD
      FUNCTION_SUBTYPE:  BYTE;
      FUNCTION_TYPE   :  BYTE;
      LR_COL          :  BYTE;
      LR_ROW          :  BYTE;
      UL_COL          :  BYTE;
      UL_ROW          :  BYTE;
      PREV_CORDS      :  AREA_POINT;
    END;

  BEGIN
    NEW( OLD_AREA );
    DEFINE_AREA_BUFFER.FUNCTION_SUBTYPE := 1;
    DEFINE_AREA_BUFFER.FUNCTION_TYPE := 0;
    DEFINE_AREA_BUFFER.LR_COL := INLR_COL;
    DEFINE_AREA_BUFFER.LR_ROW := INLR_ROW;
    DEFINE_AREA_BUFFER.UL_COL := INUL_COL;
    DEFINE_AREA_BUFFER.UL_ROW := INUL_ROW;
    DEFINE_AREA_BUFFER.PREV_CORDS := OLD_AREA;
    IO_CTL( DEFINE_AREA_BUFFER, 10);
    DEFINE_AREA := OLD_AREA;
  END; { DEFINE_AREA }


{***********************************************************************
* This function writes a single row (or part of a row) in the workspace.*
* Unlike Write Area, THIS INTRINSIC IGNORES THE AREA BOUNDS SET BY DEFINE
* AREA.  If the position and length of the data are defined such that the
* right workspace boundary is violated that portion of the data exceeding
* the boundary is ignored.  No line wrap occurs.                          *
*    INROW & INCOL  Define the position that the data will be written     *
*    BUFF_LENGTH    Defines the length of each of the following buffers  *
*    EN_BUFFER      Is a buffer with the enhancement data for each        *
*                    display position.  The enhancement characters are   *
*                    same as used in the escape programing i.e. A for    *
*                    blinking, B for inverse video, C for blinking and   *
*                    inverse video, ect.                                  *
*    CHS_BUFFER     Is the buffer with the character set code for display
*                    position.  There are 5 character sets.  The codes   *
*                    are as follows:     @ = Normal Roman                 *
*                                        A = Line Drawing                 *
*                                        B = Bold Face Roman              *
*                                        C = Italic Roman                 *
*                                        D = Math                         *
*                                    Space = No Change                    *
*    INBUFFER      The data to be displayed                               *
**************************************************************************}

PROCEDURE WRITE_LINE( INROW, INCOL, BUFF_LENGTH: INTEGER;
                      VAR EN_BUFFER, CHS_BUFFER, INBUFFER );
  VAR
    WRITE_LINE_BUFFER  : RECORD
      FUNCTION_SUBTYPE : BYTE;
      FUNCTION_TYPE    : BYTE;
      COL              : BYTE;
      ROW              : BYTE;
      BUFFER_LENGTH    : INTEGER;
      ENH_BUFFER_OFS   : INTEGER;
      ENH_BUFFER_SEG   : INTEGER;
      CHR_BUFFER_OFS   : INTEGER;
      CHR_BUFFER_SEG   : INTEGER;
      DATA_BUFFER_OFS  : INTEGER;
      DATA_BUFFER_SEG  : INTEGER;
    END;

  BEGIN
    WRITE_LINE_BUFFER.FUNCTION_TYPE := 0;
    WRITE_LINE_BUFFER.FUNCTION_SUBTYPE := 7;
    WRITE_LINE_BUFFER.ENH_BUFFER_SEG := SEG(EN_BUFFER);
    WRITE_LINE_BUFFER.ENH_BUFFER_OFS := OFS(EN_BUFFER);
    WRITE_LINE_BUFFER.CHR_BUFFER_SEG := SEG(CHS_BUFFER);
    WRITE_LINE_BUFFER.CHR_BUFFER_OFS := OFS(CHS_BUFFER);
    WRITE_LINE_BUFFER.DATA_BUFFER_SEG := SEG(INBUFFER);
    WRITE_LINE_BUFFER.DATA_BUFFER_OFS := OFS(INBUFFER);
    WRITE_LINE_BUFFER.BUFFER_LENGTH := BUFF_LENGTH;
    WRITE_LINE_BUFFER.ROW := INROW;
    WRITE_LINE_BUFFER.COL := INCOL;
    IO_CTL( WRITE_LINE_BUFFER, 18 );
  END; { WRITE_LINE }
```

```
{**********************************************************************
* This procedure Writes data into the area defined by define area     *
* See WRITE_LINE                                                      *
**********************************************************************}

PROCEDURE WRITE_AREA( DATA_LENGTH: INTEGER;
                      VAR ENH_BUFFER, CHS_BUFFER, IN_BUFFER );

  VAR
    WRITE_AREA_BUFFER          : RECORD
      FUNCTION_SUBTYPE         : BYTE;
      FUNCTION_TYPE            : BYTE;
      BUFFER_LENGTH            : INTEGER;
      ENH_BUFFER_OFS           : INTEGER;
      ENH_BUFFER_SEG           : INTEGER;
      CHR_BUFFER_OFS           : INTEGER;
      CHR_BUFFER_SEG           : INTEGER;
      DATA_BUFFER_OFS          : INTEGER;
      DATA_BUFFER_SEG          : INTEGER;
    END;

  BEGIN
    WRITE_AREA_BUFFER.FUNCTION_TYPE := 0;
    WRITE_AREA_BUFFER.FUNCTION_SUBTYPE := 2;
    WRITE_AREA_BUFFER.BUFFER_LENGTH := DATA_LENGTH;
    WRITE_AREA_BUFFER.ENH_BUFFER_OFS := OFS( ENH_BUFFER );
    WRITE_AREA_BUFFER.ENH_BUFFER_SEG := SEG( ENH_BUFFER );
    WRITE_AREA_BUFFER.CHR_BUFFER_OFS := OFS( CHS_BUFFER );
    WRITE_AREA_BUFFER.CHR_BUFFER_SEG := SEG( CHS_BUFFER );
    WRITE_AREA_BUFFER.DATA_BUFFER_OFS := OFS( IN_BUFFER );
    WRITE_AREA_BUFFER.DATA_BUFFER_SEG := SEG( IN_BUFFER );
    IO_CTL( WRITE_AREA_BUFFER, 16 );
  END; { WRITE_AREA }


{***************************************************************
* This procedure clears the area defined by the last DEFINE_AREA *
****************************************************************}

PROCEDURE CLEAR_AREA;

  VAR
    CLEAR_AREA_BUFFER          : RECORD
      FUNCTION_SUBTYPE         : BYTE;
      FUNCTION_TYPE            : BYTE;
    END;

  BEGIN
    CLEAR_AREA_BUFFER.FUNCTION_TYPE := 0;
    CLEAR_AREA_BUFFER.FUNCTION_SUBTYPE := 3;
    IO_CTL( CLEAR_AREA_BUFFER, 2 );
  END;
```

```
{*****************************************************************************
* This procedure sets the enhancement for the current area defined by the
* last DEFINE_AREA.  See WRITE_LINE for enhancement types.                   *
*****************************************************************************}

PROCEDURE ENHANCE_AREA( ENHANCE_TYPE: BYTE );

  VAR
    ENHANCE_AREA_BUF          : RECORD
      FUNCTION_SUBTYPE        : BYTE;
      FUNCTION_TYPE           : BYTE;
      FILL_BYTE               : BYTE;
      ENHANCE_BYTE            : BYTE;
    END;

  BEGIN
    ENHANCE_AREA_BUF.FUNCTION_TYPE := 0;
    ENHANCE_AREA_BUF.FUNCTION_SUBTYPE := 4;
    ENHANCE_AREA_BUF.ENHANCE_BYTE := ENHANCE_TYPE;
    IO_CTL( ENHANCE_AREA_BUF, 4 );
  END;


{*****************************************************************
* This procedure is used to input the contents of the area to the *
* program.  See WRITE_LINE.                                        *
*****************************************************************}

PROCEDURE READ_AREA( DATA_LENGTH: INTEGER;
                     VAR ENH_BUFFER, CHS_BUFFER, IN_BUFFER );

  VAR
    READ_AREA_BUFFER          : RECORD
      FUNCTION_SUBTYPE        : BYTE;
      FUNCTION_TYPE           : BYTE;
      BUFFER_LENGTH           : INTEGER;
      ENH_BUFFER_OFS          : INTEGER;
      ENH_BUFFER_SEG          : INTEGER;
      CHR_BUFFER_OFS          : INTEGER;
      CHR_BUFFER_SEG          : INTEGER;
      DATA_BUFFER_OFS         : INTEGER;
      DATA_BUFFER_SEG         : INTEGER;
    END;

  BEGIN
    READ_AREA_BUFFER.FUNCTION_TYPE := 0;
    READ_AREA_BUFFER.FUNCTION_SUBTYPE := 5;
    READ_AREA_BUFFER.BUFFER_LENGTH := DATA_LENGTH;
    READ_AREA_BUFFER.ENH_BUFFER_OFS := OFS( ENH_BUFFER );
    READ_AREA_BUFFER.ENH_BUFFER_SEG := SEG( ENH_BUFFER );
    READ_AREA_BUFFER.CHR_BUFFER_OFS := OFS( CHS_BUFFER );
    READ_AREA_BUFFER.CHR_BUFFER_SEG := SEG( CHS_BUFFER );
    READ_AREA_BUFFER.DATA_BUFFER_OFS := OFS( IN_BUFFER );
    READ_AREA_BUFFER.DATA_BUFFER_SEG := SEG( IN_BUFFER );
    IO_CTL( READ_AREA_BUFFER, 16 );
  END; { READ_AREA }


{*************************************************************************
* This procedure shifts the data in the pre-defined display area.        *
* Enhancments and character set are shifted with the ASCII data.  Data   *
* shifted off an edge of the update area is lost.                        *
*   For DIRECTION the following is used: 0 = Up                          *
*                                        1 = Down                        *
*                                        2 = Left                        *
*                                        3 = Right                       *
*   The DISTANCE is the number of Rows or Columns to be shifted          *
*   The BUFFERs are used in the remaining unshifted area. See WRITE_LINE
*************************************************************************}

PROCEDURE SHIFT_AREA( DIRECTION, DISTANCE, DATA_LENGTH: INTEGER;
                      VAR ENH_BUFFER, CHS_BUFFER, IN_BUFFER );

  VAR
    SHIFT_AREA_BUFFER        : RECORD
      FUNCTION_SUBTYPE       : BYTE;
      FUNCTION_TYPE          : BYTE;
      BUFFER_LENGTH          : INTEGER;
      ENH_BUFFER_OFS         : INTEGER;
      ENH_BUFFER_SEG         : INTEGER;
      CHR_BUFFER_OFS         : INTEGER;
      CHR_BUFFER_SEG         : INTEGER;
      DATA_BUFFER_OFS        : INTEGER;
      DATA_BUFFER_SEG        : INTEGER;
      DIST                   : BYTE;
      DIRECT                 : BYTE;
    END;

  BEGIN
    SHIFT_AREA_BUFFER.FUNCTION_TYPE := 0;
    SHIFT_AREA_BUFFER.FUNCTION_SUBTYPE := 6;
    SHIFT_AREA_BUFFER.DIST := DISTANCE;
    SHIFT_AREA_BUFFER.DIRECT := DIRECTION;
    SHIFT_AREA_BUFFER.BUFFER_LENGTH := DATA_LENGTH;
    SHIFT_AREA_BUFFER.ENH_BUFFER_OFS := OFS( ENH_BUFFER );
    SHIFT_AREA_BUFFER.ENH_BUFFER_SEG := SEG( ENH_BUFFER );
    SHIFT_AREA_BUFFER.CHR_BUFFER_OFS := OFS( CHS_BUFFER );
    SHIFT_AREA_BUFFER.CHR_BUFFER_SEG := SEG( CHS_BUFFER );
    SHIFT_AREA_BUFFER.DATA_BUFFER_OFS := OFS( IN_BUFFER );
    SHIFT_AREA_BUFFER.DATA_BUFFER_SEG := SEG( IN_BUFFER );
    IO_CTL( SHIFT_AREA_BUFFER, 18 );
  END; { SHIFT_AREA }


{**********************************************************************
* This procedure is the equivalent of the 2 Char Escape Sequences     *
* Any operation characters are valid except for those that return     *
* data.  For example if you use a INCODE of 'H' the cursor will home *
* up.  See the 'HP 150 Terminal Users Guide - Appendix A' for a list *
* valid codes.                                                        *
***********************************************************************}

PROCEDURE TWO_CHAR_SEQ( INCODE: CHAR );

  VAR
    TWO_CHAR_SEQ_BUFFER: RECORD
      FUNCTION_SUBTYPE : BYTE;
      FUNCTION_TYPE    : BYTE;
      OP_CODE          : CHAR;
    END;

  BEGIN
    TWO_CHAR_SEQ_BUFFER.FUNCTION_TYPE := 0;
    TWO_CHAR_SEQ_BUFFER.FUNCTION_SUBTYPE := 16;
    TWO_CHAR_SEQ_BUFFER.OP_CODE := INCODE;
    IO_CTL( TWO_CHAR_SEQ_BUFFER, 3 );
  END;


{**********************************************************************
* This procedure will return the absolute cursor position            *
**********************************************************************}

PROCEDURE CURSOR_SENSE_ABS( VAR CURS_ROW, CURS_COL: INTEGER );

  VAR
    CURSOR_BUFF        : RECORD
      FUNCTION_SUBTYPE : BYTE;
      FUNCTION_TYPE    : BYTE;
      CURSOR_POS_OFS   : INTEGER;
      CURSOR_POS_SEG   : INTEGER;
    END;

    CURSOR_POS         : RECORD
      CURSOR_COL       : INTEGER;
      CURSOR_ROW       : INTEGER;
    END;

  BEGIN
    CURSOR_BUFF.FUNCTION_TYPE := 0;
    CURSOR_BUFF.FUNCTION_SUBTYPE := 19;
    CURSOR_BUFF.CURSOR_POS_SEG := SEG( CURSOR_POS );
    CURSOR_BUFF.CURSOR_POS_OFS := OFS( CURSOR_POS );
    IO_CTL( CURSOR_BUFF, 6 );
    CURS_ROW := CURSOR_POS.CURSOR_ROW;
    CURS_COL := CURSOR_POS.CURSOR_COL;
  END;


{**********************************************************************
* This procedure will return the relative cursor position            *
**********************************************************************}
```

```
PROCEDURE CURSOR_SENSE_REL( VAR CURS_ROW, CURS_COL: INTEGER );

  VAR
    CURSOR_BUFF         : RECORD
      FUNCTION_SUBTYPE : BYTE;
      FUNCTION_TYPE    : BYTE;
      CURSOR_POS_OFS   : INTEGER;
      CURSOR_POS_SEG   : INTEGER;
    END;

    CURSOR_POS          : RECORD
      CURSOR_COL       : INTEGER;
      CURSOR_ROW       : INTEGER;
    END;

  BEGIN
    CURSOR_BUFF.FUNCTION_TYPE := 0;
    CURSOR_BUFF.FUNCTION_SUBTYPE := 20;
    CURSOR_BUFF.CURSOR_POS_SEG := SEG( CURSOR_POS );
    CURSOR_BUFF.CURSOR_POS_OFS := OFS( CURSOR_POS );
    IO_CTL( CURSOR_BUFF, 6 );
    CURS_ROW := CURSOR_POS.CURSOR_ROW;
    CURS_COL := CURSOR_POS.CURSOR_COL;
  END;


{*******************************************************************
* This procedure is used to set the alpha cursor type.  A CURS_TYPE *
* of 0 = Underscore and 1 = Inverse Cell.                           *
*******************************************************************}

PROCEDURE SET_CURSOR_TYPE( CURS_TYPE: BYTE );

  VAR
    SET_CURSOR_BUFF      : RECORD
      FUNCTION_SUBTYPE   : BYTE;
      FUNCTION_TYPE      : BYTE;
      CURSOR_TYPE        : BYTE;
      FILL_BYTE          : BYTE;
    END;

  BEGIN
    SET_CURSOR_BUFF.FUNCTION_TYPE := 0;
    SET_CURSOR_BUFF.FUNCTION_SUBTYPE := 21;
    SET_CURSOR_BUFF.CURSOR_TYPE := CURS_TYPE;
    IO_CTL( SET_CURSOR_BUFF, 4 );
  END;


  TYPE CHAR_BUFF = ARRAY[1..40] OF CHAR;
  TYPE BUILD_SCREEN = ARRAY[0..45] OF CHAR_BUFF;
  TYPE SCREEN = ARRAY[0..23,0..79] OF CHAR;
  TYPE BLOCK = ARRAY[1..3,1..10] OF CHAR;
```

```
VAR
  CHAR_BUFFER:       BUILD_SCREEN;
  CHAR_TYPE:         BUILD_SCREEN;
  CHAR_ENHANCE:      BUILD_SCREEN;
  OLD_AREA:          AREA_POINT;
  SCREEN_AREA:       SCREEN ABSOLUTE CHAR_BUFFER;
  SCREEN_TYPE:       SCREEN ABSOLUTE CHAR_TYPE;
  SCREEN_ENHANCE:    SCREEN ABSOLUTE CHAR_ENHANCE;
  DAY_BLOCK:         BLOCK;
  DAY_OF_WEEK:       BYTE;
  DAY_OF_MONTH:      BYTE;
  WEEK_OF_MONTH:     BYTE;
  DAY_HOLD:          STRING[2];
  TPOINT:            BYTE;

PROCEDURE FILL_DATE_BLOCK( BLOCK_ROW, BLOCK_COL: BYTE;
                           BLOCK_DATA: BLOCK );

  VAR X, Y: INTEGER;

  BEGIN
    FOR X := 1 TO 10 DO BEGIN
      FOR Y := 1 TO 3 DO BEGIN
        SCREEN_AREA[(((BLOCK_ROW * 4) - 2) + Y),
                    (((BLOCK_COL * 11) - 11) + X)] :=
         BLOCK_DATA[ Y, X ];
      END;
    END;
  END;

PROCEDURE DISPLAY_DATE_BLOCK( BLOCK_ROW, BLOCK_COL: BYTE );

  BEGIN
    WRITE_LINE(((BLOCK_ROW * 4) - 1), ((BLOCK_COL * 11) - 10), 10,
     SCREEN_ENHANCE[((BLOCK_ROW * 4) - 1), ((BLOCK_COL * 11) - 10)],
     SCREEN_TYPE[((BLOCK_ROW * 4) - 1), ((BLOCK_COL * 11) - 10)],
     SCREEN_AREA[((BLOCK_ROW * 4) - 1), ((BLOCK_COL * 11) - 10)]);
    WRITE_LINE((BLOCK_ROW * 4), ((BLOCK_COL * 11) - 10), 10,
     SCREEN_ENHANCE[(BLOCK_ROW * 4), ((BLOCK_COL * 11) - 10)],
     SCREEN_TYPE[(BLOCK_ROW * 4), ((BLOCK_COL * 11) - 10)],
     SCREEN_AREA[(BLOCK_ROW * 4), ((BLOCK_COL * 11) - 10)]);
    WRITE_LINE(((BLOCK_ROW * 4) + 1), ((BLOCK_COL * 11) - 10), 10,
     SCREEN_ENHANCE[((BLOCK_ROW * 4) + 1), ((BLOCK_COL * 11) - 10)],
     SCREEN_TYPE[((BLOCK_ROW * 4) + 1), ((BLOCK_COL * 11) - 10)],
     SCREEN_AREA[((BLOCK_ROW * 4) + 1), ((BLOCK_COL * 11) - 10)]);
  END;


BEGIN
  CHAR_BUFFER[0]  := '                                      JU';
  CHAR_BUFFER[1]  := 'NE                                      ';
  CHAR_ENHANCE[0] := '                                     BBB';
  CHAR_ENHANCE[1] := 'BBB                                     ';
  CHAR_TYPE[0]    := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
```

```
CHAR_TYPE[1]      := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_BUFFER[2]    := '     SUN        MON        TUS        WE';
CHAR_ENHANCE[2]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[2]      := 'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB';
CHAR_BUFFER[3]    := 'D        THU        FRI        SAT      ';
CHAR_ENHANCE[3]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[3]      := 'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB';
CHAR_BUFFER[4]    := 'Q;;;;;;;;;;#;;;;;;;;;;#;;;;;;;;;;#;;;;;;';
CHAR_ENHANCE[4]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[4]      := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[5]    := ';;;;#;;;;;;;;;;#;;;;;;;;;;#;;;;;;;;;;W  ';
CHAR_ENHANCE[5]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[5]      := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[6]    := ':          .          .          .      ';
CHAR_ENHANCE[6]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[6]      := 'A          A          A          A      ';
CHAR_BUFFER[7]    := '    .          .          .          :  ';
CHAR_ENHANCE[7]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[7]      := '    A          A          A          A  ';
CHAR_BUFFER[8]    := ':          .          .          .      ';
CHAR_ENHANCE[8]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[8]      := 'A          A          A          A      ';
CHAR_BUFFER[9]    := '    .          .          .          :  ';
CHAR_ENHANCE[9]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[9]      := '    A          A          A          A  ';
CHAR_BUFFER[10]    := ':          .          .          .      ';
CHAR_ENHANCE[10]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[10]      := 'A          A          A          A      ';
CHAR_BUFFER[11]    := '    .          .          .          :  ';
CHAR_ENHANCE[11]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[11]      := '    A          A          A          A  ';
CHAR_BUFFER[12]    := '!,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,';
CHAR_ENHANCE[12]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[12]      := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[13]    := ',,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,"  ';
CHAR_ENHANCE[13]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[13]      := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  ';
CHAR_BUFFER[14]    := ':          .          .          .      ';
CHAR_ENHANCE[14]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[14]      := 'A          A          A          A      ';
CHAR_BUFFER[15]    := '    .          .          .          :  ';
CHAR_ENHANCE[15]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[15]      := '    A          A          A          A  ';
CHAR_BUFFER[16]    := ':          .          .          .      ';
CHAR_ENHANCE[16]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[16]      := 'A          A          A          A      ';
CHAR_BUFFER[17]    := '    .          .          .          :  ';
CHAR_ENHANCE[17]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[17]      := '    A          A          A          A  ';
CHAR_BUFFER[18]    := ':          .          .          .      ';
CHAR_ENHANCE[18]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[18]      := 'A          A          A          A      ';
CHAR_BUFFER[19]    := '    .          .          .          :  ';
CHAR_ENHANCE[19]   := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
```

```
CHAR_TYPE[19]     := '    A          A          A          A  ';
CHAR_BUFFER[20]   := '!,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,';
CHAR_ENHANCE[20]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[20]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[21]   := ',,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,"  ';
CHAR_ENHANCE[21]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[21]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  ';
CHAR_BUFFER[22]   := ':          .          .          .      ';
CHAR_ENHANCE[22]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[22]     := 'A          A          A          A      ';
CHAR_BUFFER[23]   := '    .          .          .          :  ';
CHAR_ENHANCE[23]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[23]     := '    A          A          A          A  ';
CHAR_BUFFER[24]   := ':          .          .          .      ';
CHAR_ENHANCE[24]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[24]     := 'A          A          A          A      ';
CHAR_BUFFER[25]   := '    .          .          .          :  ';
CHAR_ENHANCE[25]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[25]     := '    A          A          A          A  ';
CHAR_BUFFER[26]   := ':          .          .          .      ';
CHAR_ENHANCE[26]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[26]     := 'A          A          A          A      ';
CHAR_BUFFER[27]   := '    .          .          .          :  ';
CHAR_ENHANCE[27]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[27]     := '    A          A          A          A  ';
CHAR_BUFFER[28]   := '!,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,';
CHAR_ENHANCE[28]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[28]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[29]   := ',,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,"  ';
CHAR_ENHANCE[29]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[29]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  ';
CHAR_BUFFER[30]   := ':          .          .          .      ';
CHAR_ENHANCE[30]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[30]     := 'A          A          A          A      ';
CHAR_BUFFER[31]   := '    .          .          .          :  ';
CHAR_ENHANCE[31]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[31]     := '    A          A          A          A  ';
CHAR_BUFFER[32]   := ':          .          .          .      ';
CHAR_ENHANCE[32]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[32]     := 'A          A          A          A      ';
CHAR_BUFFER[33]   := '    .          .          .          :  ';
CHAR_ENHANCE[33]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[33]     := '    A          A          A          A  ';
CHAR_BUFFER[34]   := ':          .          .          .      ';
CHAR_ENHANCE[34]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[34]     := 'A          A          A          A      ';
CHAR_BUFFER[35]   := '    .          .          .          :  ';
CHAR_ENHANCE[35]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[35]     := '    A          A          A          A  ';
CHAR_BUFFER[36]   := '!,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,';
CHAR_ENHANCE[36]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
CHAR_TYPE[36]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
CHAR_BUFFER[37]   := ',,,,/,,,,,,,,,,/,,,,,,,,,,/,,,,,,,,,,"  ';
CHAR_ENHANCE[37]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
```

```
  CHAR_TYPE[37]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  ';
  CHAR_BUFFER[38]   := ':          .          .          .      ';
  CHAR_ENHANCE[38]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[38]     := 'A          A          A          A      ';
  CHAR_BUFFER[39]   := '    .          .          .          :  ';
  CHAR_ENHANCE[39]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[39]     := '    A          A          A          A  ';
  CHAR_BUFFER[40]   := ':          .          .          .      ';
  CHAR_ENHANCE[40]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[40]     := 'A          A          A          A      ';
  CHAR_BUFFER[41]   := '    .          .          .          :  ';
  CHAR_ENHANCE[41]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[41]     := '    A          A          A          A  ';
  CHAR_BUFFER[42]   := ':          .          .          .      ';
  CHAR_ENHANCE[42]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[42]     := 'A          A          A          A      ';
  CHAR_BUFFER[43]   := '    .          .          .          :  ';
  CHAR_ENHANCE[43]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[43]     := '    A          A          A          A  ';
  CHAR_BUFFER[44]   := 'A;;;;;;;;;;$;;;;;;;;;;$;;;;;;;;;;$;;;;;;';
  CHAR_ENHANCE[44]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[44]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
  CHAR_BUFFER[45]   := ';;;;$;;;;;;;;;;$;;;;;;;;;;$;;;;;;;;;;S  ';
  CHAR_ENHANCE[45]  := '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@';
  CHAR_TYPE[45]     := 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  ';
  TWO_CHAR_SEQ('H');
  TWO_CHAR_SEQ('J');
  NEW( OLD_AREA );
  OLD_AREA := DEFINE_AREA( 24, 79, 0, 0 );
  DAY_BLOCK[ 1 ] := '           ';
  DAY_BLOCK[ 2 ] := '           ';
  DAY_ BLOCK[ 3 ] := '           ';
  DAY_OF_MONTH := 1;
  FOR WEEK_OF_MONTH := 1 TO 5 DO BEGIN
    FOR DAY_OF_WEEK := 1 TO 7 DO BEGIN
      STR( DAY_OF_MONTH:2, DAY_HOLD );
      FOR TPOINT := 1 TO 2 DO BEGIN
        DAY_BLOCK[ 1, TPOINT + 8 ] := DAY_HOLD[ TPOINT ];
      END;
      DAY_OF_MONTH := DAY_OF_MONTH + 1;
      FILL_DATE_BLOCK( WEEK_OF_MONTH, DAY_OF_WEEK, DAY_BLOCK );
    END;
  END;
  WRITE_AREA( 1840, CHAR_ENHANCE, CHAR_TYPE, CHAR_BUFFER );
  DAY_BLOCK[ 2 ] := '***********';
  DAY_BLOCK[ 1 ] := '         XX';
  FILL_DATE_BLOCK( 2, 3, DAY_BLOCK );
  DISPLAY_DATE_BLOCK( 2, 3 );
END.
```

3051. Cooperating Processes in an Information Network

Peter Gerstenhaber
Systems Specialist
CMS Ltd.
11 Masad St.
Tel Aviv, Isreal

OVERVIEW:

Traditional system design, and program design techniques are single threaded. Most programs are designed to accept and validate data, and then perform the desired request before accepting the next input. In single machine configurations this may lead to systems which meet the desired response time requirements, but suffer when additional users are added. In larger systems, or those which require more than one machine to satisfy a user's request this may lead to a poor or unpredictable response time.

The purpose of this paper is to present a design of a multi-cpu system which is in use today utilizing the above mentioned features. This system provides the user with a uniform response time across many machines with varying loads. The primary purpose of this choice of design was not to decrease response time, but to decompose the project into managable programs which could be worked on independently. As a by-product, the application provides a high level interface to data bases spread across multiple machines.

INTRODUCTION:

Trends in system design have typically lagged behind computer developments by as much as a decade or more. In the 60's and 70's, system design was mostly dictated by machine architecture. Early systems were based completely on batch input, where many batch jobs had the same characteristics. Batch jobs typically were composed of the edit (verification of data format), validation (data consistency), update, and reporting phases with the option of reformatting for the next batch job (since everything sometime or other gets fed into the general ledger application). Dependent upon complexity, any of these steps consisted of any number of programs, each with it's own exception report(s), and sorting for proper input sequence. The main point of this description, is to note the sequentiality of the data flow. Today, we don't work as much with big batches of transactions, but instead with single transactions performed on-line, usually by a single program. This program must accept and edit the data, validate for correctness (existance of part-number, location-code, etc), update the necessary data base(s), and report sucess or failure. The same program, or an additional one usually performs the data reporting function. As with batch jobs, these events are performed in a sequential order, without any overlap. Essentially, the outcome is that we are now performing on-line in the same manner as batch jobs were performed. The difference being that transactions from different users must use locking to avoid inconsistencies when updating.

This sequentiality exists in spite of the fact that MPE is a multi-programming operating system (i.e. that it shares the system's resources in an orderly fashion among the processes competing for these functions), and was not the first of this kind. Concurrency has been built into the operating systems of many machines, but not much further. There isn't an industry standard language available which allows concurrent activities within a single application system, without specifically imbedding within the application calls to system routines. This results in many application systems being implemented by large sequential programs which utilize alot of stack space, open many files/data bases, and may not exhibit the desired performance or modularity. This makes program maintance difficult or next to impossible due to the size of the program, stack limitations, and/or subsystem data areas. MPE V/E with the micro-code allowing table expansion overcomes the previous limitations of program size, but does not address stack limitations in the least. Programs which use V/PLUS may need an additional 8-10k words DL-DB (or more), leaving precious little left for programs which declare all their variables globally.

THE PROBLEM:

An application problem requires multiple IMAGE data bases, and is logically distributed accross many systems. The problem is to design and implement a system to handle multiple data bases, and multiple CPU's in a consistent fashion with a high level interface which may be accessed from TRANSACT/3000, COBOLII/3000, and PASCAL/3000. It is required that one site be designated as the "central" system, and that there be several "local" applications which are logically connected. A local system may only communicate with the central system, while the central system may communicate with any or all local systems. This is a typical star topology. Systems may share processors, so that several "local" application systems may share the same physical machinery with each other, and/or with the "central" system. There should not be any restrictions imposed as to the combination of application systems per physical processor. DSN/DS and DSN/X.25 will be used for machine interconnection. The central system is to have a consolidated data base which consists of the sum of all the data bases of each local system. This data base is to be kept sychronized, and should be designed to be fault tolerant (i.e. so that a transaction applied to a local data base will be applied to the central data base without regard to system/data communication problems). The central site is to control all data communications, and allow each local system access to the central data base. Users at any site may query either the data base at their local site, or the central data base. Users may update any data base for which they have access. There a class of user who may update data at a different local system then the one they reside on; while a second class may update data only at their local site. The system should provide complete auditing information, so that a log file may be queried by data, date, time, terminal, transaction type, and/or location.

PROCESS HANDLING, A DIFFERENT PERSPECTIVE:

In designing a large multi-cpu system to solve this problem, we did not reach any of the MPE limitations. Instead, our first problem was establishing a locking heirarchy for a multi-data base environment. This was easily worked out, but was a precursor of problems which were not anticipated (such as data consistency in a multi-cpu, multi-data base environment). As the system was being designed and implemented, it was noticed that there were transactions which did not meet the severe response time requirements. Of the entire set of transactions, there were several which meant massive changes to the data bases, and could conceivable lock the data bases for minutes (1-3) while they were being processed. Clearly these transactions had an adverse effect upon overall system response, and an extremely pronounced effect on response time for all terminals using this particular application. With this in mind, it was decided to decouple all data base updates from the rest of the application. In essence, the on-line application only performed a reporting function and data entry. This data entry was written to a transaction file which was processed by a background process concurrently.

The advantages of implementing a system based on background updates are numerous. It was meant to decouple terminal response time from data base updates, and gave on-line users an extremely fast response. As a by-product, there were several advantages to this approach. Since each transaction was atomic, there was no need for IMAGE logging. Each transaction could conceivably translate into many DBUPDATE/PUT/DELETE's. Auditing was easily accomplished at the transaction level, and was almost a by-product of this implementation. Auditing information was available with user name, terminal, program, form file, and time with the rest of the transaction. A complete record of all transactions could be formatted by any or all or the above criteria. If timing information was included within the transaction, response time and system utilization could be easily calculated. This would have the advantage of calculating application response time (as opposed to system response time) which is meaningful to any system analyst.

It could be argued that this method serializes all updates to all data bases envolved unnecessarily. This is true, but on the other hand since a single transaction is a related piece of information, it is localized. At the time this application was written, there was no need for a seperate process for each data base (as in a data base server approach), but this could have easily been implemented. The data base server approach keeps intact data integrity at the transaction level, but allows transactions which apply to multiple data bases to proceed in parallel. This left the on-line program running without any locking, and therefore without delays due to multi-user access in an update mode.

Additional benefits were derived from this design methodology which proved important in a development environment. Since a program was restricted in scope, it was smaller in size, with a corresponding smaller stack. There was less overhead since there was one stack which was performing data base maintance. Additionally, in a multi-cpu system, there was one flow of transactions throughout the entire system as opposed to each on-line terminal performing remote data base queries and updates. This feature will be examined in closer detail after a discussion on the features of MPE which facilitate a distributed environment. Basically this method of functional decomposition of a problem is similar to utilizing pipes in UNIX. Unlike UNIX pipes, however, MPE provides for sychronization of processes which are not for the same user, or necessarily on the same machine.

It may be argued that this design methodology has problems with users performing concurrent updates. In most applications, where weak locking is used, if two users update the same record, the user whose updates are performed last is the user whose data remains in the data base. With the above methodology, this remains the same. The major drawbacks are that this doesn't allow for strong locking, and by it's very design this is a non-deterministic system with respect to time (i.e. the time an update will take place cannot be determined, but it is guaranteed to happen). If this is acceptable, then this is possibly a good design alternative.

THE TOOLS:

The CREATEPROCESS intrinsic is notable mainly because of it's benefits over the CREATE/ACTIVATE intrinsics. Being able to create a process and redirect $STDIN/$STDLIST has the advantage that an application program written in COBOLII, or PASCAL (or SPL for that matter) using standard language features as READ and WRITE can access a file other than the standard input/output files of the session, and moreover this is performed transparent to the application. The CREATEPROCESS intrinsic sets up the environment for a program to be run, complete with it's own data stack. This program may optionally be scheduled to run at the same time. When this program is run (ACTIVATEd), it competes for the CPU and all other resources with all other processes (running programs) in an asychronous fashion. Asychronously means that this process and all other processes will use the machines resources in an indeterminate fashion, without being able to determine the order of completion.

IPC greatly enhances the file system with two file types and many different types of access possible. Circular (CIR) files are log files which hold the last N records (where N is at most the file limit). The behavior of these files is similar to a sequential file until they reach the end of file. As records are written, they are appended to the end of the file. When the file is full, additional records are written, and the oldest records (the ones at the front of the file) are deleted to make room for these new records. Note that the beginning of file changes as additional records are written after the file is full. This is also known as a "wrap-around" file for this reason. Circular files have the restriction that they can only be opened for one type of access at a time (i.e. open for either read access, or write access to multiple users, but not both).

Interprocess communication (IPC) also includes message (MSG) files which permits multiple user processes to communicate via the file system. A message file is a FIFO (first in, first out) queue of messages (records). Records are added to the end of the queue via a WRITE, and received and deleted via a READ. In each file open to a message file, the process must specify either READ (the process is identified as a READER) or WRITE access (the process is identified as a WRITER). This is described in the MPE INTRINSICS MANUAL as a "unidirectional flow of information". If a process needs to update a message file, it must open the message file twice; as both a READER and a WRITER. Any message which a READER receives is deleted from the file automatically by the file system. There is a special access allowing the entire message file to be read without destroying it's contents. This access is call COPY access, but the process requesting this type of access needs EXCLUSIVE access to the message file. This is essential for development when the contents of the file are to be viewed before processing (or a copy made via FCOPY). Typical applications for message files are with many producers of messages (WRITERS) and one consumer (READER), although the file system allows many combinations dependent upon the access options. The defaults for a message file allow global multi-access and exclusive access (one READER, and one WRITER). If a message file is opened with SEMIexclusive access, only one READER is allowed, but there may be multiple WRITERS. SHR access allows any number of READERS and WRITERS at the same time. With the file open, the allowable structure of READERS/WRITERS is determined. A message file allows either MULTI or GMULTI access, with MULTI restricting access to processes within the same job/session. GMULTI access allows unrelated processes to communicate with one another only by utilizing the file system. This implies that for an IPC application with cooperation among processes, there is no longer a need for process creation and the restriction that these processes be related. Because of this, message files are an extremely friendly and easy method of interprocess communication.

As an added benefit, message files use GMULTI access, which means that there is one control block and set of buffers for all accessors. In it's implementation, these buffers are only written to disc when necessary. When the READER can keep consuming the messages at the rate they are being produced by the WRITERS, there is no disc i/o, but inter-data segment memory transfers. Therefore, the impact on overall performance is negligable, and with access the extra data segment containing the control block will remain in memory.

MSG FILE FEATURES:

Message files are by nature variable in record length. This permits users to send and receive variable length messages via the file system. into the Unrelated processes are queued automatically, and therefore sychronized. A READER will wait when reading an empty message file, as a WRITER will wait when writing to a full message file. A variable time-out interval may be set which limits the length of time it may take for a request to be serviced before it is aborted. With this facility, there is no need for a process to suspend indefinitely. Note that checking the end- of-file is not sufficient since in a multiprogramming environment other processes maybe performing the same check and then writing, before the first process which checked the "end-of-file". Non-destructive reads allow a READER to preview the head of the queue without destroying this message. Multiple READERS may read the same message non-destructively, but only one destructive READER will delete the message. Care needs to be taken with multiple READERs using this feature. This is the only type of file which permits non-privileged mode nowait I/O. Nowait I/O allows a process to perform a write (or a read) and check later within the program for it's completion. Typical nowait I/O is also unbuffered which implies the need of privileged mode, but all I/O to a message file uses a sharable control block and buffers. Because of this, nowait I/O for message files does not require the use of privileged mode. Software interrupts are supported which allow a trap facility to be invoked upon completion of an event (typically a read). This allows messages to be received asychronously and handled out of the main-line code similar to a control-y trap. A trap procedure is entered when I/O completes and the interrupt is generated. After the trap routine completes, control transfers to the next sequential instruction in the mainline code. Nowait I/O and software interrupts are features available on local message files only, and cannot be used with remote file access. All other features are available with both local and remote file access. Most importantly, message files provide the means of interprocess communication and sychronization without having to resort to artifical conventions.

THE SOLUTION:

The solution to this problem was quite simple. Once the data base update program had been written, it needed a small change to run at the central site, and the local site. With this intact, there was left the data communications to feed into this system. This was split into two programs, one to handle operator commands and responses, and the other for the actual data communications. Between the two, more than one method of interprocess communications and sychronization was used.

Data communications was performed completely via the file system. This program was written in PASCAL/3000 using standard system intrinsics, and requiring IA,BA,and PH capabilities. This process has access to only one "local" site, and is run from the central site. The following description will be a simplification since the application requires six message files and two sets of data bases (one for performing application queries, and the second to determine routing information of other data communication handlers). The input transaction file could be either a remote file (the usual case) or a local file (in this case, the "local" site was on the same physical machine as the "central" site; locations were to be transparent to this application). Basically all this program did was to read it's input transaction, process the request, and then delete the transaction from the input file. For those interested in performance with remote file access, the message was deleted by reading it via FREAD with a zero length. This transaction could be any one of several types. The usual transaction was reporting an update which already took place at the local site and needed to be applied to the central site. In this case, the transaction was routed to the output message file. This message file was being written to by each of the data comm handlers, of which there was one for each location. This is the input file for the process which updates the central machine's data base. If the transaction was from a local site requesting that a different local site's data base be updated, this request was routed to the data comm handler for the requested site via a write to a message file. A local user may query the central data base via a transaction. This query may be qualified with selection criteria, ranges, and/or boolean operations. In this manner the query only returns to the local machine those entries which meet all criteria. If remote queries had been allowed, all data meeting the search criteria with IMAGE would have to be transmitted to the local machine, regardless of the selection criteria. With this method, only males between the ages of 30 and 50, with blue eyes, living in New York City, with the name of Zacharia Smith would be retrieved across the communications lines as opposed to all Smiths on the system. Additionally, a request might ask for all information about the selected entries (i.e. all detail information directly related to Zacharia Smith's master entry). A message file is opened up to the process making this request dynamically, and closed each time a request is received for a different process. In this fashion, the message file stays open if a terminal process makes multiple query requests one after the other (which is the usual case).

A different message file handles requests from the central machine. Requests in this file could be from users of the central machine, or other data comm processes trying to route information to the local site to which this process is connected. This information is handled asychronously. Whenever a record is written to this file, it is immediately transfered to the local site. This is handled by using soft interrupts, and nowait I/O on the message file. An example of using soft interrupts on a message file is included later.

A parent process was used to create and monitor activity to all locations. This process used a data base which had the necessary information to create the environment for each data communication process (specifically file equations, run parm, run info string, and file redirection). Communications between the control process and each data comm process was via MAIL intrinsics. This was chosen because in all cases, only one word of information needed to be exchanged, and this information could only be made use of at specific points in the program, so there was no need to trap, but to check. This alleviated the overhead of the file system, and substituted memory transfers (of one word) from stack to PPCT to stack. This process checks for the existance of each of it's sons, and warns the operator when a son process terminates abnormally.

THE TRANSFER PROCESS IN FULL:

```
[Console Msgs]    [Central DB]    [Config DB]    [Transactions]
             \              |      |             /
              \             |      |            /
               \            |      |           /
                \           |      |          /
                 \          |      |         /
                  \         |      |        /
                   \        |      |       /
                    \       |      |      /
                     \      |      |     /
                      \     |      |    /
                       \    |      |   /
                        \   |      |  /
                         \  |      | /
                          \ |      |/
[Interrupt In] ----------- (Transfer) ---------- [Msg Routing]
                          / |      |\
                         /  |      | \
                        /   |      |  \
                       /    |      |   \
                      /     |      |    \
                     /      |      |     \
                    /       |      |      \
                   /        |      |       \
                  /         |      |        \
                 /          |      |         \
                /           |      |          \
               /            |      |           \
              /             |      |            \
       [Mass Add]        [Transactions]    [Query Responses]
```

Of these files, the "Query Response" and "Msg Routing" are files which are switched dynamically to meet the demands.

SYSTEM OVERVIEW:

```
                      Central Machine

                    [Central Data Base]
                   /        |         \
                  /         |          \
                 /          |           \
                /           |            \
               /       (DB Update)        \
              /             |              \
             /              |               \
            /               |                \
           /                |                 \
          /   [Consolidated Transaction File]  \
         /        /                     \       \
        /        /                       \       \
       /        /                         \       \
      (Transfer)           ...           (Transfer)
      /       |                            |      \
     /        |                            |       \
    /         |                            |        \
[Interrupt In]|                            |  [Interrupt In]
              |                            |
              |                            |
              |                            |
++++++++++++++++++  Data Communications  ++++++++++++++++++
              |              +             |
 Local        |              +             |       Local
Machine       |              +             |      Machine
  "A"   [Transaction]        +   [Transaction]      "B"
              |              +        |
              |              +        |
              |              +        |
[Local DB] < < < (DB Update) + (DB Update) > > > [Local DB]
    |              |         +     |                 |
    |              |         +     |                 |
    |              |         +     |                 |
    |       [Transaction]    + [Transaction]         |
    |             |          +      |                |
    |             |          +      |                |
    |             |          +      |                |
   (Terminal Server)         +    (Terminal Server)
          .                  +          .
          .                  +          .
          .                  +          .
   (Terminal Server)         +    (Terminal Server)
```

Note that processes are enclosed in parenthesis, while files and data bases are enclosed in square brackets. A transfer process is created per local system, while terminal server processes communicate with each terminal connected to this application.

EXAMPLE:

Initialization Code:

```
     asych_file_io := FOpen('file-name');
     IF (CCODE <> good_cc)
     THEN
         fatal_file_error(asych_file_io);
     FControl (asych_file_io
              ,enable_soft_interrupts
              ,WADDRESS(soft_interrupt_handler
              );
     IF (CCODE <> good_cc)
     THEN
         fatal_file_error(asych_file_io);
     FRead(asych_file_io); (* Note that this assembles the I/O
          request but does not block us, or transfer any data
          even if data is available *)
     IF (CCODE <> good_cc)
     THEN
         fatal_file_error(asych_file_io);
```

Interrupt Routine:

```
(****************************************************************)
(*                                                              *)
(*                   Soft_Interrupt_Handler                     *)
(*                                                              *)
(*   This procedure handles soft interrupts.  This is enabled   *)
(*   for only the Central_to_Local file.  When a record is      *)
(*   written to this file, this program will be interrupted     *)
(*   asychronously (i.e. anywhere it is), and control will be   *)
(*   transfered to this procedure by the Operating System,      *)
(*   transparent to the logic of the program.  We will then     *)
(*   read this record, write it to the remote machine, and re-  *)
(*   establish a no-wait i/o request for this file.             *)
(****************************************************************)
```

```
PROCEDURE soft_interrupt_handler(f_num:  small_integer);
  PROCEDURE FIntExit(state:  BOOLEAN);
            EXTERNAL SPL VARIABLE; (* Intrinsic Definition *)
(****************************************************************)
(*                                                              *)
(*                        Read_Write                            *)
(*                                                              *)
(*   This procedure handles the I/O for the soft interrupts.    *)
(*   It is in a seperate procedure so that the variables        *)
(*   will not be nested if there is a backup on this message    *)
(*   file (note that the FIntExit re-enables interrupts         *)
(*   before this procedure can finish.  Theoretically, many     *)
(*   interrupts may  be nested and we could stack overflow.     *)
(*   By moving the variables into this procedure they are       *)
(*   allocated and deallocated before the FIntExit so that      *)
(*   we will not have a stack overflow.                         *)
(****************************************************************)
  PROCEDURE read_write(f_num:  small_integer);
    VAR
      io_rec:      STD_REC;
      io_rec_len:  small_integer;
    PROCEDURE IoDontWait;  INTRINSIC;
    BEGIN (* read_write *)
       IoDontWait(f_num, io_rec, io_rec_len);
       IF (CCODE <> good_cc)
       THEN
           fatal_file_error(f_num);
       FWrite(remote_dbupd_fnum, io_rec, -io_rec_len, 0);
       IF (CCODE <> good_cc)
       THEN
           fatal_file_error(remote_dbupd_fnum);
       FRead(f_num, io_rec, -SIZEOF(io_rec));
    END;  (* read_write *)


  BEGIN (* soft_interrupt_handler *)
     IF (f_num <> asych_file_io)
     THEN
         fatal_file_error(f_num);
     read_write(f_num);
     FIntExit;
  END;  (* soft_interrupt_handler *)
```

Note that a nested procedure is used to accept the data into a dynamic buffer. If this were incorporated into the trap procedure, there could be a problem with multiple requests interrupting as soon as the FINTEXIT is executed, and before the procedure is exited. This problem would be with stack space being acquired for each activation of this procedure.

SUMMARY:

IPC is a valuable tool which allows the system designer to make use of "piping". Applications may be functionally decomposed to allow different processes to perform different functions.

IPC is more powerful than "piping" with the addition of nowait I/O, and soft interrupt processing. This enables the application program to have more than one input file, and handle each input file seperately, without having to check each input file. The example showed only one file with soft interrupt processing enabled, but any number of message files can utilize both nowait I/O and soft interrupt processing.

IPC isn't dependent upon process handling, but can be used in any environment where process communication and sychronization is required.

IPC can be used for intermachine data transfers in an easy manner. There is no longer the need for complicated programming techniques to allow an extremely flexible, and transparent interconnection. A high degree of data system integrity may be achieved in a system utilizing multiple data bases across multiple systems.

IPC is available with any language which supports READs and WRITEs. This means that RPG, COBOLII, PASCAL, TRANSACT, etc have transparent access to IPC.

IPC is an integral facility used in the development and application of a major distributed system which has been described in this paper. Without this facility, this project would have taken more than TEN times the effort to accomplish the data communications necessary.

3052. Creating Custom Applications On The HP3000

Marv Miller
Computer Systems Divison
Hewlett Packard

SUMMARY

One of the most time consuming functions of an application system's programming team is program maintenance. Quite often this time is related to the kinds of activity that can be greatly reduced or even eliminated by taking full advantage of the integration of Transact/3000 and the dictionary.

These kinds of changes are those where new input forms are being added, existing input forms changed or deleted, new data elements are being added, and other already existing elements are being changed in size or eliminated.

This paper follows through examples of localizing a program, that is, making it independent of the screen name and the number of screens that execute the same code, independent of the screen contents, and providing user exits for additional processing. The changes are always made to the dictionary. The changes become effective in the program by simply recompiling the program in order to pull in the new dictionary definitions.

INTRODUCTION

An application system can be divided into at least two parts. The first part is made up of the data that is needed by the system processing logic. This data or these data elements are critical to the proper functioning of the system. For example, a manufacturing system no doubt has an element called PART-NUMBER which is a critical part of practically all system transactions.

A typical application may have several critical data elements. It is fair to say that a localizable application can not allow critical fields to be deleted from the application. Application programs rely upon these fields to be present in transactions.

However, a localizable application should allow these elements to be changed in size. It should also allow the physical placement of these elements within an input form to be changed.

The other part of an application is made up of non-critical elements. Many of these elements may be supplied as a part of the original application, if for no other reason than the typical generic application has these data elements. Other non-critical elements may be added by the particular user of the application.

A localizable application should allow non-critical elements to be added, deleted, changed in size, and changed in physical placement within an input form.

Also, generic transactions should be localizable. A generic transaction is defined here to be a transaction that provides a basic function such as adding a customer or updating a customer. For example, one organization may be responsible for adding new customers to the database, but several organizations need to be able to update portions of the customer data. Each organization should be provided with a form which only accesses the data they need. The same program logic that provides customer update capability should be able to handle any number of these variations.

Finally, a localizable application should allow logic to be added to handle such things as: special field edits for any of the transaction's data elements, data calculations, etc.

The following discussion explains how Transact can achieve this level of localization. The objective is to write an application program such that if the application is changed as described above, the program is not modified. The changes need to be recorded in the dictionary, the program recompiled to make the changes known to it, the VPLUS forms file modified to reflect the changes, and possibly the database unloaded and reloaded if its structure has been modified.

A GENERIC TRANSACTION

First, a simple transaction that only applies to one dataset. This demonstrates all of the concepts to be achieved through localization. Later an example is given of a generic transaction that applies to several datasets, in order to demonstrate the general case of how to write generic code.

The discussion starts with a transaction to update information for a customer. Breaking this transaction into the two parts discussed above, the critical element in this transaction is CUST-NO. The non-critical elements are: NAME, STREET-ADDR, CITY-STATE, and ZIP-CODE.

The VPLUS form used for this function is VCUSTOMER.

```
**********************************************************************
*vcustomer               customer data                              *
*                                                                   *
*                       number  [____]                              *
*                                                                   *
*                       name    [____________________]              *
*                                                                   *
*                       address [____________________]              *
*                                                                   *
*                  city,state   [____________________]              *
*                                                                   *
*                       zipcode [______]                            *
*                                                                   *
**********************************************************************
```

The CUSTOMER dataset definition is:

```
   FILE: CUSTOMER     TYPE: MASTER

 ELEMENT           PROPERTIES:

   CUST-NO          I+(4,0,2)
   NAME             X (20,0,20)
   STREET-ADDR      X (20,0,20)
   CITY-STATE       X (20,0,20)
   ZIPCODE          X (6,0,6)
```

The following program illustrates how a transaction to update a customer might be written without allowing for any localization. This program will be expanded to illustrate most of the localization concepts.

```
system custfm,base=orders,vpls=formfile;
list(auto) customer;

level;
get(form) vcustomer,init;

set(key) list (cust-no);
get customer,list=(@);
put(form) vcustomer,window=("update? - f1=yes, f2=no");
get(form) vcustomer,f1(autoread)=modify-f1
                   ,f2=modify-f2;

modify-f1:

  update customer,list=(@);

modify-f2:

  end;


```

```
22     exit:
23
24       exit;
```

The program uses the same form to initially input the customer to be updated (line 5), display the current data for the customer (line 9), and input the new data for the customer (line 10).

REARRANGING THE SCREEN

Perhaps the easiest form of localization is to rearrange the order of elements on the screen. The program form specification does not include any element ordering information. This is controlled thru the dictionary. Thus, this localization can be accomplished by modifying the form using FORMSPEC, changing the element sequence on the form definition in the dictionary, and recompiling the program using TRANCOMP.

The same program could then handle input from a form such as this:

```
*********************************************************************
*vcustomer                  customer data                           *
*                                                                   *
*                                                                   *
*      name    [_____________________]      number [____]           *
*                                                                   *
*      address [_____________________]                              *
*                                                                   *
*  city,state  [_____________________]                              *
*                                                                   *
*      zipcode [______]                                             *
*                                                                   *
*                                                                   *
*********************************************************************
```

Changing the screen definition in the dictionary might go something like this:

```
**********************************************************************
:run dictdbm.pub.sys

DICTIONARY/3000  HP32244A.02.01  - (C) Hewlett-Packard Co. 1984

 PASSWORD FOR DICT.PUB>

FORMS ENTRY(Y/N)?>

> show file
                 FILE vcustomer

FILE                 TYPE: RESPONSIBILITY:
 VCUSTOMER            FORM
```

```
    ELEMENT(ALIAS):             PROPERTIES:         ELEMENT(PRIMARY):
     CUST-NO                     I+(4,0,2)           CUST-NO
     NAME                        X (20,0,20)         NAME
     STREET-ADDR                 X (20,0,20)         STREET-ADDR
     CITY-STATE                  X (20,0,20)         CITY-STATE
     ZIPCODE                     X (6,0,6)           ZIPCODE

> resequence file
                FILE vcustomer

             ELEMENT name
        NEW POSITION cust-no

             ELEMENT
> show file
                FILE vcustomer

FILE                    TYPE: RESPONSIBILITY:
 VCUSTOMER               FORM

    ELEMENT(ALIAS):             PROPERTIES:         ELEMENT(PRIMARY):
     NAME                        X (20,0,20)         NAME
     CUST-NO                     I+(4,0,2)           CUST-NO
     STREET-ADDR                 X (20,0,20)         STREET-ADDR
     CITY-STATE                  X (20,0,20)         CITY-STATE
     ZIPCODE                     X (6,0,6)           ZIPCODE
***********************************************************************
```

SCREEN INDEPENDENCE

The above program can be modified to illustrate implementing generic transaction code that can handle multiple screen formats. The program could have been written this way to start with. It is not a program modification that must be made every time a new form is added.

The following program provides generic update customer capability and is form independent. That is, the program has no idea of which data elements exist on a form, nor does it know how many possible different forms may be used to update a customer.

```
***********************************************************************
system custup,base=orders
             ,vpls=formfile
             ,file=formxref;
define(item) menuname x(16):
             fkey     9(2):
             screen   x(16):
             lastkey i(4);
list menuname:
     lastkey;

data menuname; <<to simulate transfer of control to this
                      subroutine>>
```

```
<<
****************************************************************
Subroutine: to update customer information

input: menuname - contains the name of the screen to be displayed

output: none
****************************************************************
>>
level;
list fkey:
     screen;
list(auto) customer;
get(form) (menuname),init
                    ,window=(" ")
                    ,fkey=lastkey
                    ,autoread;
if (lastkey) = 0
  then perform modify
else
do
  set(match) list (menuname);
  let (fkey) = (lastkey);
  set(match) list (fkey);
  get(serial) formxref,list=(menuname,fkey,screen);
  reset(option) match;
  perform modify;
  doend;
end;
modify:

  set(key) list (cust-no);
  get customer,list=(@);
  put(form) (screen),window=("update? - f1=yes, f2=no");
  get(form) (screen),f1(autoread)=modify-f1
                    ,f2=modify-f2;

modify-f1:

  update customer,list=(@);

modify-f2:

  end;
************************************************************************
```

It uses Transact's indirect referencing capability for forms. Notice that all verbs which reference a screen name do not actually specify the screen name. Each verb specifies the name of an element which contains the name of the screen to be referenced.

The program sets up a menu driven customer update capability such as the following series of screens depict.

```
**************************************************************************
*custupdatemm            customer update main menu                       *
*                                                                        *
*                                                                        *
*              enter customer number      [1    ]                        *
*                                                                        *
*                                                                        *
*                      f1 -  marketing         (custupdate1)             *
*                                                                        *
*                      f2 -  finance           (custupdate2)             *
*                                                                        *
*                      f3 -  accounts payable  (custupdate3)             *
*                                                                        *
*              **********************************************            *
*                                  or                                    *
*                                                                        *
*                      enter screen name [                ]              *
*                                                                        *
*                                                                        *
**************************************************************************
*market-* finance* accounts*        *        *        *        *  exit   *
*ing    *        * payable *        *        *        *        *         *
**************************************************************************

**************************************************************************
*update1                 marketing customer update                       *
*                                                                        *
*                                                                        *
*                                                                        *
*                       customer number  [1   ]                          *
*                                                                        *
*                       name             [name of customer 1  ]          *
*                                                                        *
*                                                                        *
*update?   f1=yes, f2=no                                                 *
**************************************************************************

**************************************************************************
*custupdate2                  finance customer update                    *
*                                                                        *
*                                                                        *
*                            customer number [1   ]                      *
*                                                                        *
*                            zip code        [12345 ]                    *
*                                                                        *
*                                                                        *
*update?   f1=yes, f2=no                                                 *
**************************************************************************
```

```
*******************************************************************************
*custupdate3               accounts payable customer update                   *
*                                                                             *
*                          customer number [1   ]                             *
*                                                                             *
*                                     name [name of customer 1  ]             *
*                                                                             *
*                                  address [108 Lincoln Ave.    ]             *
*                                                                             *
*                               city,state [So. Bend, Ind.      ]             *
*                                                                             *
*                                  zipcode [12345 ]                           *
*                                                                             *
*                                                                             *
*update? - f1=yes, f2=no                                                      *
*******************************************************************************
```

There are many ways to implement a form independent program. The above is just one illustration. Probably the key to this implementation is the MPE file called FORMXREF which provides the indirection needed to establish form independence.

The content of FORMXREF is as follows:

```
**************************************
*MENUNAME:         FKEY: SCREEN:     *
* ----------------------------------*
* CUSTUPDATEMM      1     CUSTUPDATE1*
* CUSTUPDATEMM      2     CUSTUPDATE2*
* CUSTUPDATEMM      3     CUSTUPDATE3*
**************************************
```

MENUNAME and FKEY are the index into the file specifying the menu that the user is currently working with and the function key just pressed by the user to indicate the next screen to go to. SCREEN contains the name of the next data entry screen to use.

When this program begins, the element menuname contains the name of the menu that controls its functionality. Line 11 simulates this by prompting for the menu name. When prompted for the menu name, CUSTUPDATEMM was typed in.

The menu allows the user to specify the next screen in either of two ways. The name of the screen can be entered in the box titled enter screen name. The <ENTER> key enters this data and lines 30 and 31 detect this and perform the update routine. Or, the screen can be indicated via a function key. If this way is chosen, the file FORMXREF is accessed to determine the screen name to be used by the modify routine. Lines 34 thru 39 accomplish this.

The cross reference file has a record for each function key of each screen that defines the name of the screen to use when that function key is pressed. In our example, if the user presses <f1>, then screen CUSTUPDATE1 is used.

Another form for updating a customer could now be designed and used by this program merely by recompiling the program. Of course, the form would have to be designed in FORMSPEC and defined in the data dictionary first.

ADDING, DELETING, CHANGING ELEMENTS

The following illustrates a major change to the CUSTOMER dataset, expanding the size of CUST-NO from 4 to 6 digits long, deleting the ZIPCODE field and adding a new field called AREA. The important point for Transact is that the program only needs to be recompiled to incorporate the new structure.

First, the form file is modified, changing all the forms that reference the customer data.

CUSTUPDATEMM is changed to reflect the 6 digit customer number.

```
************************************************************************
*custupdatemm            customer update main menu                     *
*                                                                      *
*                                                                      *
*              enter customer number      [      ]                     *
*                                                                      *
*                                                                      *
*                      f1 -  marketing         (custupdate1)           *
*                                                                      *
*                      f2 -  finance           (custupdate2)           *
*                                                                      *
*                      f3 -  accounts payable  (custupdate3)           *
*                                                                      *
*              ********************************************            *
*                                 or                                   *
*                                                                      *
*                      enter screen name [               ]             *
*                                                                      *
*                                                                      *
************************************************************************
*market-* finance* accounts*       *        *        *        *  exit   *
*ing    *        * payable *       *        *        *        *         *
************************************************************************
```

CUSTUPDATE1 is changed to reflect the 6 digit customer number.

```
***********************************************************************
*custupdate1                   marketing customer update              *
*                                                                     *
*                                                                     *
*                                                                     *
*                        customer number  [      ]                    *
*                                                                     *
*                        name             [                  ]        *
*                                                                     *
*                                                                     *
*                                                                     *
***********************************************************************
```

CUSTUPDATE2 is changed to reflect the 6 digit customer number, delete of ZIPCODE, and addition of AREA, because finance needs to be able to update this new code.

```
***********************************************************************
*custupdate2                   finance customer update                *
*                                                                     *
*                                                                     *
*                              customer number [     ]                *
*                                                                     *
*                                         area [     ]                *
*                                                                     *
*                                                                     *
*                                                                     *
***********************************************************************
```

CUSTUPDATE3 is changed to reflect the 6 digit customer number, delete of ZIPCODE, and addition of AREA, because accounts payable needs to be able to update all fields.

```
***********************************************************************
*custupdate3                  accounts payable customer update        *
*                                                                     *
*                             customer number [     ]                 *
*                                                                     *
*                                        name [                  ]    *
*                                                                     *
*                                     address [                  ]    *
*                                                                     *
*                                  city,state [                  ]    *
*                                                                     *
*                                        area [     ]                 *
*                                                                     *
*                                                                     *
*                                                                     *
***********************************************************************
```

These changes as well as the database changes are recorded in the dictionary using DICTDBM.

```
**********************************************************************
:run dictdbm.pub.sys
DICTIONARY/3000  HP32244A.02.01  - (C) Hewlett-Packard Co. 1984
 PASSWORD FOR DICT.PUB>
FORMS ENTRY(Y/N)?>

> modify element
               ELEMENT cust-no
EDIT DESCRIPTION(Y/N)?> n
ELEMENT                TYPE: SIZE: DEC: LENGTH: COUNT:    RESPONSIBILITY:
 CUST-NO                i+    4     0    4       1
      LONG NAME:
   HEADING TEXT:
     ENTRY TEXT:
      EDIT MASK:
MEASUREMENT UNITS:
 BLANK WHEN ZERO: NO

                  TYPE i+
                  SIZE 6
               DECIMAL !

> create element
               ELEMENT area
             LONG NAME
                  TYPE x
                  SIZE 6
  STORAGE LENGTH(6)    !

> delete file
                  FILE custupdate3
               ELEMENT zipcode
ENTRY DELETED
               ELEMENT

> add file
                  FILE custupdate3
               ELEMENT area
         ELEMENT ALIAS
          FIELD NUMBER
           DESCRIPTION
               ELEMENT

> delete file
                  FILE custupdate2
               ELEMENT zipcode
ENTRY DELETED
               ELEMENT

> add file
                  FILE custupdate2
               ELEMENT area
         ELEMENT ALIAS
          FIELD NUMBER
```

```
             DESCRIPTION
                 ELEMENT

  > delete file
                    FILE customer
                 ELEMENT zipcode
  ENTRY DELETED
                 ELEMENT

  > add file
                    FILE customer
                 ELEMENT area
           ELEMENT ALIAS
             DESCRIPTION
                 ELEMENT

  > exit

  END OF PROGRAM
  :
  **********************************************************************
```

Next the database is unloaded using DICTDBU.

```
  **********************************************************************
  run dictdbu.pub.sys
  DICTIONARY/3000 DB UNLOADER      HP32244A.02.01 - (C) Hewlett-Packard
  STORE FILE> mpestore
  LIST FILE>
  BASE> orders
  BASE PASSWORD>
  MODE> 1
  UNLOAD AUTOMATIC MASTER SETS(N/Y)?>
  UNLOAD DETAIL SETS BY CHAIN(Y/N)?>
  UNLOAD EDIT(N/Y)?>
  PROCESSING SETS
  CUSTOMER         M:2/100
   2 ENTRIES UNLOADED IN <1 CPU-SEC
  PARTS            M:2/100
   2 ENTRIES UNLOADED IN <1 CPU-SEC
  ORDER            A:2/100
   AUTO NOT UNLOADED
  INVENTORY        D:3/108
   3 ENTRIES UNLOADED IN <1 CPU-SEC
  ORDERHEAD        D:2/112
   2 ENTRIES UNLOADED IN <1 CPU-SEC
  ORDERLINE        D:3/100
   3 ENTRIES UNLOADED IN <1 CPU-SEC
  UNLOAD COMPLETED
  END OF PROGRAM
  :
************************************************************************
```

Then the current database is purged using DBUTIL.

```
*************************************************************************
  run dbutil.pub.sys
  HP32215B.04.61 IMAGE/3000: DBUTIL  (C) COPYRIGHT HEWLETT-PACKARD
  >>pur orders
   Data base has been PURGED.
  >>exit

  END OF PROGRAM
  :
*************************************************************************
```

Then the new database root file is created using DICTDBC.

```
*************************************************************************
  run dictdbc.pub.sys
  DICTIONARY/3000 DB CREATOR       HP32244A.02.01 - (C) Hewlett-Packard
  DICTIONARY PASSWORD>
  BASE> orders
  CONTROL LINE>
  SCHEMA FILE>
  LISTING FILE>
  APPLY SECURITY JUST TO SET LEVEL(N/Y)?>
  SCHEMA GENERATION
  DBSCHEMA PROCESSOR
  PAGE 1      HEWLETT-PACKARD 32215B.04.50 IMAGE/3000: DBSCHEMA
         TUE, MAY 14, 1985,  9:21 AM  (C) HEWLETT-PACKARD CO. 1978
  BEGIN DATA BASE ORDERS;
  PASSWORDS:
  ITEMS:
      AREA,              X6        ;
      CITY-STATE,        X20       ;
      CUST-NO,           I2        ;
      DESCRIPTION,       X20       ;
      LINE-NO,           X2        ;
      LOCATION,          X4        ;
      NAME,              X20       ;
      ORDER-DATE,        X6        ;
      ORDER-NO,          X8        ;
      ORDER-STATUS,      X2        ;
      PART-NUMBER,       X8        ;
      QUANTITY,          I2        ;
      STREET-ADDR,       X20       ;
  SETS:
  NAME:   CUSTOMER,       MANUAL    ;
  ENTRY:  CUST-NO         (1),
         NAME,
         STREET-ADDR,
         CITY-STATE,
         AREA;
  CAPACITY: 100;
  NAME:   PARTS,          MANUAL    ;
  ENTRY:  PART-NUMBER     (2),
         DESCRIPTION;
  CAPACITY: 100;
```

```
NAME:   ORDER,            AUTOMATIC ;
ENTRY:  ORDER-NO         (2);
CAPACITY: 100;
NAME:   INVENTORY,        DETAIL    ;
ENTRY:  PART-NUMBER      ( PARTS          ),
       LOCATION,
       QUANTITY;
CAPACITY: 100;
NAME:   ORDERHEAD,        DETAIL    ;
ENTRY:  ORDER-NO         ( ORDER          ),
       CUST-NO         ( CUSTOMER       ),
       ORDER-STATUS,
       ORDER-DATE;
CAPACITY: 100;
NAME:   ORDERLINE,        DETAIL    ;
ENTRY:  ORDER-NO         ( ORDER          ),
       LINE-NO,
       PART-NUMBER     ( PARTS          ),
       QUANTITY;
CAPACITY: 100;
END.
DATA SET      TYPE  FLD  PT  ENTR  MED  CAPACITY  BLK  BLK   DISC
  NAME              CNT  CT  LGTH  REC            FAC  LGTH  SPACE
CUSTOMER         M  5    1   36    46    100       11   507   44
PARTS            M  2    2   14    29    100       13   378   27
ORDER            A  1    2   4     19    100       20   382   18
INVENTORY        D  3    1   8     12    120       40   483   16
ORDERHEAD        D  4    2   11    19    100       20   382   18
ORDERLINE        D  4    2   11    19    100       20   382   18
                          TOTAL DISC SECTORS INCLUDING ROOT: 152
NUMBER OF ERROR MESSAGES: 0
ITEM NAME COUNT: 13     DATA SET COUNT: 6
ROOT LENGTH: 587     BUFFER LENGTH: 507     TRAILER LENGTH: 256
ROOT FILE ORDERS CREATED.
END OF PROGRAM
:
*****************************************************************
```

The new database is created using DBUTIL.

```
*************************************************************************
  run dbutil.pub.sys
  HP32215B.04.61 IMAGE/3000: DBUTIL  (C) COPYRIGHT HEWLETT-PACKARD
  >>cre orders
    Data base ORDERS has been CREATED.
  >>exit
  END OF PROGRAM
  :
*************************************************************************
```

The database is reloaded using DICTDBL.

```
*************************************************************************
  run dictdbl.pub.sys
  DICTIONARY/3000 DB LOADER       HP32244A.02.01 - (C) Hewlett-Packard
  STORE FILE> mpestore
  LIST FILE>
   BASE: ORDERS.CUSTOMIZ.MILLER
  RUN MODE(LOAD/EDIT/SHOW/EXIT)>
  NEW BASE NAME>
  BASE PASSWORD>
  MODE>
  FAST I/O(Y/N)?>
  CUSTOMER            M 2/100
  ZIPCODE           ITEM NOT FOUND, NEW ITEM NAME>   <cr> = not reloaded
   2 ENTRIES LOADED IN <1 CPU-SEC
  PARTS               M 2/100
   2 ENTRIES LOADED IN <1 CPU-SEC
  INVENTORY           D 3/120
   3 ENTRIES LOADED IN <1 CPU-SEC
  ORDERHEAD           D 2/100
   2 ENTRIES LOADED IN <1 CPU-SEC
  ORDERLINE           D 3/100
   3 ENTRIES LOADED IN <1 CPU-SEC
  LOAD COMPLETED
  END OF PROGRAM
  :
*************************************************************************
```

Finally, the Transact program is recompiled and the new application is implemented.

```
*************************************************************************
  run trancomp.pub.sys
  TRANSACT/3000 COMPILER   HP32247A.02.02 - (C) Hewlett-Packard Co. 1984
  SOURCE FILE> custup
  LIST FILE>
  CONTROL> nolist
  TRANSACT/3000 COMPILER  A.02.02 : TUE, MAY 14, 1985,  9:32 AM COMPILED L
   OF FILE CUSTUP.CUSTOMIZ.MILLER            PAGE 1
  COMPILING WITH OPTIONS: CODE,DICT,ERRS
  CODE FILE STATUS: REPLACED
  0 COMPILATION ERRORS
  PROCESSOR TIME=00:00:08
  END OF PROGRAM
  :
*************************************************************************
```

DICTDBU and DICTDBL can not handle all types of element changes. For example, numeric ascii will not be converted correctly by these utilities. This is because IMAGE does not have a data type corresponding to numeric ascii. DICTDBC creates an element defined as numeric ascii as an alphanumeric element of type X. Thus, if CUST-NO were being changed from 9(4) to 9(6), DICTDBU would unload it as X(4). DICTDBL would reload it as X(6) causing the new field to be left justified with two spaces inserted on the right. Transact would no longer be able to interpret the field as numeric.

Thus,when writing a custom application, avoid using data type 9 or write a utility to convert data after DICTDBU has run and before DICTDBL has run.

USER EXITS

In general, there are two types of application development environments. First there are application software companies who build application solutions to sell to other companies. Second, there are companies who build application solutions for use internally. Localization is attractive to both environments. Ignoring the fact that the software is sold in one case, both types of environments have a similar structure. There is a central group responsible for maintaining and enhancing the core system. There are one or more user organizations who accept the basic system functionality, but who have unique needs from the other users. Until these needs become the common needs of the majority of the system users, the central group typically resists adding the functionality to the core system.

If, however, the central group provides ways in which the individual users can modify the system to include the functionality they need without destroying the functionality of the core system, then both groups become winners.

The concepts discussed earlier, plus the concept of user exits provide this capability.

It is probably much easier for the software engineers developing software to sell to build in user exits, since they are acutely aware of the many unique demands their customers make.

It is no doubt much more difficult for a software engineer building internal software to distinguish between capabilities that should be a part of the core system versus capabilities that should be extensions or localization of the core system, since his users are also internal.

Providing the capability for user exits and when to provide the capability for user exits is up to the designer of the core system. For example, a designer may only want to allow user intervention after data has been entered. Another designer may want to allow user intervention before and after data entry, as well as before and after database update.

Naturally, the more a designer provides this capability, the better the possibility that the user can solve his unique problem outside of the core system.

The example below illustrates one way to implement user exits within Transact. It implements a structure that allows the user to do some processing just after data has been entered.

```
**************************************************************************
system custex,base=orders
             ,vpls=formfile
             ,file=formxref,exitxref;
```

```
define(item) menuname x(16):
             fkey     9(2):
             screen   x(16):
             lastkey i(4):
             userexit-prog x(6):
             userexit-marker @;
list menuname:
     lastkey:
     userexit-prog;

data menuname; <<to simulate transfer of control to this
                 subroutine>>
<<
*************************************************************
Subroutine: to update customer information

input: menuname - contains the name of the screen to be displayed

output: none
*************************************************************
>>
list fkey:
     screen;
list userexit-marker;
list(auto) custupd-global;
let (yes) = 1;
let (no) = 0;
let (error) = (no);
level;
list(auto) customer;
if (error) = (no)
  then
    get(form) (menuname),init
                        ,window=(" ")
                        ,fkey=lastkey
                        ,autoread
  else
    get(form) (menuname)
                        ,fkey=lastkey
                        ,autoread;
if (lastkey) = 8
  then exit;
set(match) list (menuname);
get(serial) exitxref,list=(menuname,userexit-prog);
call (userexit-prog),data=userexit-marker;
if (error) = (yes)
  then end;
if (lastkey) = 0
  then perform modify
else
  do
  set(match) list (menuname);
  let (fkey) = (lastkey);
  set(match) list (fkey);
```

```
 58        get(serial) formxref,list=(menuname,fkey,screen);
 59        reset(option) match;
 60        perform modify;
 61        doend;
 62      end;
 63
 64      modify:
 65
 66        set(key) list (cust-no);
 67        get customer,list=(@);
 68        put(form) (screen),window=("update? - f1=yes, f2=no");
 69        get(form) (screen),f1(autoread)=modify-f1
 70                          ,f2=modify-f2;
 71
 72      modify-f1:
 73
 74        update customer,list=(@);
 75
 76      modify-f2:
 77
 78        end;
*************************************************************************
```

The user exit is established in a way similar to that used to achieve screen independence. A cross reference file is set up to contain the name of the sub-program to be called based upon the name of the screen that the program is currently processing.

The content of this cross-reference file is:

```
********************************
*MENUNAME:      USEREXIT-PROG:*
*-----------------------------*
* CUSTUPDATEMM    CU1         *
********************************
```

MENUNAME is the index into the program specifying the current menu or form. USEREXIT-PROG contains the name of the Transact sub-program to be called.

As this program illustrates, there can be some data defined within the program for its own use (lines 10 thru 26). This data could also be defined in the dictionary. There can also be global data that is of importance to both the program and the user exit program (lines 28 thru 31). This data should be defined in the dictionary in order to make coding of the user exit program easier. Included in this data are elements for handling screen data input errors (validation) and the data the user wants to add to the transaction. Finally, there is the dataset definition needed specifically for this generic transaction, also defined in the dictionary (line 33). The dictionary description of custupd-global and customer is as follows:

```
*************************************************************************
FILE                  TYPE: RESPONSIBILITY:
 CUSTUPD-GLOBAL        FORM
```

```
        ELEMENT(ALIAS):            PROPERTIES:         ELEMENT(PRIMARY):
         PASSWORD                   X (8,0,8)            PASSWORD
         ERROR                      I (4,0,2)            ERROR
         YES                        I (4,0,2)            YES
         NO                         I (4,0,2)            NO

FILE                  TYPE: RESPONSIBILITY:
 CUSTOMER              MAST

        ELEMENT(ALIAS):            PROPERTIES:         ELEMENT(PRIMARY):
         CUST-NO                    I+(6,0,4)            CUST-NO
         NAME                       X (20,0,20)          NAME
         STREET-ADDR                X (20,0,20)          STREET-ADDR
         CITY-STATE                 X (20,0,20)          CITY-STATE
         AREA                       X (6,0,6)            AREA
*************************************************************************
```

Note that the user of the system has added the element PASSWORD to the CUSTUPD-GLOBAL list. This element is not a part of the core application.

The program depends upon the existence of ERROR, YES, and NO as the way in which the sub-program indicates to the main program that an error has been detected. The main program initializes these variables in lines 29 to 31.

The program sets up a marker element which it uses to denote the point in the LIST register that the sub-program has access to (line 27 and 48).

The screen is displayed without erasing the information, if the user exit program detected an error. Otherwise an initialized screen is displayed. Lines 34 to 45 handle this.

Lines 46 to 48 implement the user exit by searching for a match on screen name in the cross-reference file. This code and file could be expanded to provide for multiple user exits during the same transaction and to make a user exit optional.

The user exit program follows. The user has decided to add a password to the customer update menu and has added the logic to his sub-program to validate the password.

```
************************************************************
system cul,vpls=formfile;
list(auto) custupd-global;
list(auto) customer;
let (error) = (no);
if (password) <> "OKAY"
  then
    do
    set(form) *,window=(password,"invalid password");
    let (error) = (yes);
    doend;
exit;
************************************************************
```

Note that the user exit program must contain the LIST definition corresponding to the main program definition that occurs after the marker item. Standardized procedures for communicating error conditions, etc. must also exist.

The modified user version of the screen CUSTUPDATEMM and the dictionary description of this screen follows:

```
************************************************************************
*custupdatemm            customer update main menu                     *
*                                                                      *
*                                                                      *
*              enter customer number      [      ]                     *
*                                                                      *
*                                                                      *
*                      f1 -  marketing        (custupdate1)            *
*                                                                      *
*                      f2 -  finance          (custupdate2)            *
*                                                                      *
*                      f3 -  accounts payable (custupdate3)            *
*                                                                      *
*              **********************************************          *
*                                or                                    *
*                                                                      *
*                      enter screen name [                ]            *
*                                                                      *
*                                                                      *
*                                                                      *
*                      password [        ]                             *
*                                                                      *
*                                                                      *
************************************************************************
*market-* finance* accounts*       *        *        *       *  exit   *
*ing     *        * payable *       *        *        *       *         *
************************************************************************

************************************************************************
  FILE                 TYPE: RESPONSIBILITY:
   CUSTUPDATEMM         FORM

        ELEMENT(ALIAS):            PROPERTIES:          ELEMENT(PRIMARY):
          CUST-NO                    I+(6,0,4)            CUST-NO
          SCREEN                     X (16,0,16)          SCREEN
          PASSWORD                   X (8,0,8)            PASSWORD
************************************************************************
```

TRANSACTIONS ACROSS MULTIPLE DATASETS

All of the above concepts are still valid even if the transaction affects multiple datasets. The following program illustrates a way to write generic code that accesses more than one dataset. This code could be expanded to include the topics previously discussed to provide form independence, user exits, etc.

```
******************************************
system addprt,base=orders
             ,vpls=formfile;
list(auto) addpart-global;
level;
list(auto) partvendors;
level;
list(auto) inventory;
level;
list(auto) parts;
get(form) addpart,init;
put parts,list=(@);
move (global-part) = (part-number);
end(level);
move (part-number) = (global-part);
put inventory,list=(@);
end(level);
move (part-number) = (global-part);
put partvendors,list=(@);
end;
******************************************
```

The generic transaction adds a new record to the parts dataset, which is the master dataset. It then adds a record to each of two detail sets, inventory and partvendors. PART-NUMBER is a critical element and is common to all three sets.

The dictionary description of the lists used by the program are as follows:

```
*************************************************************************
FILE                  TYPE: RESPONSIBILITY:
 ADDPART               FORM

        ELEMENT(ALIAS):             PROPERTIES:         ELEMENT(PRIMARY):
         PART-NUMBER                 X (8,0,8)           PART-NUMBER
         DESCRIPTION                 X (20,0,20)         DESCRIPTION
         LOCATION                    X (4,0,4)           LOCATION
         QUANTITY                    I (6,0,4)           QUANTITY
         VENDOR-CODE                 X (6,0,6)           VENDOR-CODE
         VENDOR-NAME                 X (20,0,20)         VENDOR-NAME

FILE                  TYPE: RESPONSIBILITY:
 ADDPART-GLOBAL        FORM

        ELEMENT(ALIAS):             PROPERTIES:         ELEMENT(PRIMARY):
         GLOBAL-PART                 X (8,0,8)           GLOBAL-PART

FILE                  TYPE: RESPONSIBILITY:
 INVENTORY             DETL

        ELEMENT(ALIAS):             PROPERTIES:         ELEMENT(PRIMARY):
         PART-NUMBER           *     X (8,0,8)           PART-NUMBER
                                CHAIN MASTER SET: PARTS
        LOCATION                     X (4,0,4)           LOCATION
        QUANTITY                     I (6,0,4)           QUANTITY

FILE               TYPE: RESPONSIBILITY:
 PARTS              MAST

       ELEMENT(ALIAS):            PROPERTIES:          ELEMENT(PRIMARY):
        PART-NUMBER        *       X (8,0,8)            PART-NUMBER
        DESCRIPTION                X (20,0,20)          DESCRIPTION

FILE               TYPE: RESPONSIBILITY:
 PARTVENDORS        DETL

       ELEMENT(ALIAS):            PROPERTIES:          ELEMENT(PRIMARY):
        PART-NUMBER        *       X (8,0,8)            PART-NUMBER
                               CHAIN MASTER SET: PARTS
        VENDOR-CODE                X (6,0,6)            VENDOR-CODE
        VENDOR-NAME                X (20,0,20)          VENDOR-NAME
*************************************************************************
```

Note that the global definitions for this transaction include an element called GLOBAL-PART. This element is used to store the value of PART-NUMBER between dataset updates as explained below.

The form ADDPART looks like this:

```
**************************************************************************
*addpart                     add a part                                  *
*                                                                        *
*                                                                        *
*              part number [        ]                                    *
*                                                                        *
*              description [                    ]                        *
*                                                                        *
*              location    [    ]                                        *
*                                                                        *
*              quantity    [      ]                                      *
*                                                                        *
*              vendor code [      ]                                      *
*                                                                        *
*              vendor name [                    ]                        *
*                                                                        *
*                                                                        *
*                                                                        *
**************************************************************************
```

The key to understanding how to write generic code is to understand how the VPLUS and IMAGE interface work with the LIST register.

The first thing to understand is that the LIST register can have as many definitions of an element on it as wanted. However, Transact always references the latest definition. Thus,the LIST(AUTO) for each dataset that the transaction is to access, causes three definitions of PART-NUMBER to be added to the LIST register.

The VPLUS interface with the dictionary does not require use of the LIST= option. If this is left off, Transact matches the elements that are a part of the form with the current contents of the LIST register. These elements can occur anywhere physically in the LIST register. Elements are resolved by starting at the end (most recent change) of the LIST register, and working back until the element definition is found (line 10).

The IMAGE interface through the LIST=(@), requires the element list to be contiguous. Individual elements can not be listed, since this would defeat the idea of creating custom code. Thus after updating the PARTS dataset (line 11), the value of PART-NUMBER is saved (line 12), then all of the PARTS dataset elements are removed (line 13), then the value of PART-NUMBER is restored, which is now the PART-NUMBER defined for dataset INVENTORY (line 14). A record is then added to the INVENTORY dataset.

Since PART-NUMBER has already been saved, it need not be saved again. All of the elements that belong to the INVENTORY set are removed from the LIST and then PART-NUMBER is restored, which now becomes the PART-NUMBER for the PARTVENDORS set.

This same logic can be repeated any number of times. Similar logic also handles data retrieval from different sets.

3053. RECOVERY BY DESIGN / Surviving a Disaster

James A. Depp
UPTIME
4131-A Power Inn Road
Sacramento, California 95826

Overview of the Presentation:

A.) Do I really need protection/planning? Do disasters really occur? Can it happen to me?
 - Anecdotes and experiences -
B.) Why bother? How is it going to benefit me? How do I sell the boss? Is there a payout?
C.) OK - how do I do it? What's the blueprint?
D.) Perspective - tasks/program and design considerations
 1) Resources and difficulty in replacing
 2) Equipment/Facility
  a) Options
  b) Agreements
  c) Recovery considerations
  d) Design considerations
 3) Data and Software
  a) Recovery considerations
  b) Design considerations
 4) People
 5) Communications
  a) Human
  b) Data
 6) Supplies
E.) Question and Answer

CONTINGENCY PLANNING - WHY BOTHER - WHAT ARE THE BENEFITS?

The auditors insist on it, the user asks what happens to my operation if the computer fails; the staff doesn't like to think about it - a dreaded event, not often acknowledged, but always a concern, uncertainty, fear.

HOW DO I SELL THE BOSS? - the more tangible reasons for planning; the ones that lead to top management commitment, and action?

1) Protect your business - experience shows small to medium size companies dependent on computer records may be bankrupt in one to two weeks if they can't access their information. At best they will be crippled for months.
2) Protect the company's market position and stock price since improperly communicating the loss of computing resources could cause loss of customers or adverse market reaction.
3) Reduced insurance - business failure insurance is carried by many companies - protection of the business reduces the risk and potentially the premiums.
4) Protection from legal redress - current legislation and

precedent allow stockholders to take civil action against management for failure to properly protect the business from potential threat.

5) Increased productivity and performance - close examination of any operation from a different vantage point can result in improvement. Recovery planning has resulted in reduced data base size and content, improving response time. Data communications have been reevaluated and streamlined causing less frequent attention. Operations have been improved as options were presented by the in depth survey.
6) Staff training opportunities are presented both in preparation of the recovery stategy and in performing the annual test. One operations supervisor and the operator who brought their backup tapes for a test said this was more valuable than HP coursework because they had full use and control of the system, and had to set up and check all aspects.

On a broader scale, contingency planning involves not only the data processing arena. You will find that user support and the entire business plans will follow more easily if the critical data is available on a backup computer system.

Additionally, there are on-going benefits to the information systems staff as well - briefly:

1) streamlined data center operations
2) streamlined business because it is examined
3) smooth, accurate response to minor failures
4) enhanced image within business organization
5) increased DP job satisfaction - reduced turnover

THE DISASTER RECOVERY PROGRAM -

Not just a written plan, recovery is a program which is begun at a point in time, and which continues actively into the future. The program objectives are:

1) Reduce potential losses to a level which the company can absorb without endangering the business.
2) Create a plan immune to company change and growth.
3) Balance resources between program needs and other competing development, operational and business needs.

A key part of the program is assessing the VALUE of the processing rather than the cost, and to support and protect the people who will be directly and indirectly involved in recovery.

DISASTER PLANNING: The strategy for reestablishing information processing.

Disaster planning is an audit of the present to assess what will be critically needed if disaster strikes - and to provide those things. It is a list of tasks - strategic and tactical - using the resources protected and available to recreate the processing environment. It is organization of the 'significant few' with others assisting so that the recovery is smoothly and effectively executed as quickly as possible. It is the testing of itself, routinely, to be certain that the plan is always viable.

HOW IS IT DONE? - BLUEPRINTS

Steps to building a plan:

1) Top management commitment
2) Define critical applications with the users and gain commitment.
3) Estimate the business value of the processing = cost of lost processing.
4) Determine how long an outage is tolerable
5) Determine the restoration period
6) Document requirements/assumptions with user concurrence.

Preparation:

1) Allocate staff to the planning process. You may, due to limited internal resources, seek consulting assistance. But be certain that the resulting strategy is specific to your organization and not just a document.
2) Secure storage of vital records and resources. You may already store your backup tapes offsite. There may be other items to store, including a copy of the plan.
3) Confirm that your insurance coverage is sufficient and that payment of the claim will be speedy. Also discuss the insurance company's actions if there is a disaster. In some cases a partially damaged computer system was unavailable for several days because the insurance company had not completed their assessment.
4) Assess your documentation in its current form. Are the routine operational activities in written form? Is the source code organized and well commented? The desire here is to utilize as much of the existing material as possible in the plan formation, and vice versa. The impact of staff turnover can also be minimized through this planning effort.
5) Itemize the services that will need to be supported for the applications determined to be critical. These may be specific data required, data communications, people, reports, etc.
6) Itemize the supplies which will be needed as forms, labels, etc. and don't forget such related equipment as bursters, signature writers, and the like. You may later decide not to provide some of these, but do so consciously.

Plan Elements:

Now is the time to begin the actual plan creation with its decision making and refinement activity.

1) What is required? State the requirements as already determined.
2) Where will the processing be done, and how?
3) Who will be responsible for recovery, and what aspects will they perform? What are the limits to their activity?
4) Write the plan and the documentation.
5) Monitor the plan effectiveness and people availability at least annually, revising the plan as required. For some organizations, more frequent tests will be required.

PERSPECTIVE - nuts and bolts

Effective operational procedures and advance enlistment of assistance can ensure the resource availability. Routine system backups will place the software and data offsite, safe from the disaster which destroys the processing base. Procedures for use of the tapes when they return from storage protects these resources from loss, particularly in a time of confusion. The successful recovery is dependent upon people, data, equipment, site, communications, and supplies availability. The discussion of these specifics will begin with equipment and facility because the availability of these so directly affects the others.

HARDWARE -

We recogize that equipment must be available, but where? Options include the HP office, HP quick replacement, a reciprocal, a captive hotsite, third party vendors, additional systems within the company which are underutilized routinely. -OR- you might join a recovery service which provides equipment, data center, staff, and testing - the advantage is cost and availability - a system that can be reloaded now, which is known available and has been tested, and which is located near your people, regardless of where you have to put them. And extra trained hands which don't exist routinely can be made available, as well.

Recovery Options might be ranked as follows:

	Effectiveness	Cost
-Dedicated/owned hotsite	Excellent	Very High
-Mobilized hotsite service	Excellent	Affordable
-Stationary recovery service hotsite	Good	Affordable
-HP Rush Replacement	Varies	Varies
-Third party hardware source	Varies	Varies
-Shell site	Varies	Varies
-Reciprocal agreements	Marginal	Low
-Service Bureaus	Poor	Varies
-Do Nothing	?	0

PRIOR AGREEMENTS:

Prior agreements are the only method for making hardware available because capacity must be available, or paperwork completed in order to have equipment delivered or made accessible now, rather than days later. Any other business or organization with which you deal will have their own contrary objectives when you need the system. Agreements include:

1) contracts and frequent tests of reciprocals - remember that both parties are developing and could cause the change which would interfere with recovery.
2) pre-executed purchase orders with hardware suppliers.
3) written statements, frequently reviewed, saying what service could be provided on short notice by HP or other of your own divisions.
4) Commitment of all involved to spend the money to test regularly, which means not only the cost to go test, but for the receiving facility to shrink and prepare for the test. How many of you have run a test at the local HP office of the commitment to make the machine available, let alone actually bringing your system up there? What about the local service bureau? Your reciprocal partner? Why not?

HARDWARE DESIGN CONSIDERATIONS:

In general, standardize or document carefully. HP hardware is generally well understood in interfacing, use, capability. There is nothing wrong with using foreign equipment, but the interfacing may be unusual, so be sure it is documented. It will not be in the standard HP manuals. And where specific hardware dependent patches to the operating system or programming are required, put these in job streams.

Examples include cabling to multiplexors/modems, the non-HP equipment such as microcomputers and wordprocessors where pins 4,5,6 and 8 are jumped to 20. Multipoint printers over modems/multiplexors may require pin 6 to be jumped.

Selection of peripherals, particularly for data communications, should include the recovery consideration. Consider the ramifications of antique, unique, or ultra-new hardware. If the computer is to autodial the modem, and interrogate a remote source of data, it might be preferable to use equipment with the 'AT' type command structure, since these modems are readily available, even in microcomputer stores. Where multiplexors and high speed equipment is needed, choose the company which has in place a 24 hour emergency replacement strategy - we've only located one, so far.

FACILITY:

To have ready access to hardware is only part of the successful recovery. How much planning was required when your own data center was designed and built? The equipment needs specific electrical characteristics, air conditioning, static protection, data communications connections. Your staff will need work space, lighting, water, rest room facilities, etc. It is well enough to say that the equipment can be had reasonably quickly, but will the facility be available to make the equipment usable?

FACILITY DESIGN CONSIDERATIONS:

When installing terminal wiring it would be preferable to terminate individual terminal wiring remote to the computer room, and to bring multiconductor cables into the machine. This is neater,and it gives a remote tie-in location, carefully marked for clarity. Likewise, modems can be installed 50' from the computer, giving the possibility that they might survive destruction of the system.

Power and power conditioning can likewise be protected and marked, as well as having alternate sources of power identified and marked.

SOFTWARE AND DATA -

Data must be protected both against loss and inaccessibility. It is the companion of software, which can be similarly protected. Protection hinges on data and software storage. Options include:

Dump strategy and Offsite storage
enhanced by:
Remote data logging and Shadowing

DATA DESIGN CONSIDERATIONS:

The dump strategy includes full, partial, and selective dumps. In each case there are a few watchouts:

- Do you label dump tapes? If so, stop immediately, since these cannot be read.
- Do you read the output listing to note errors, problems, and skipped files? This is not just the beginning or end of the listing.
- Do you store the dump tapes securely while they await offsite courier pickup?
- Are all files included such that program changes will necessarily be dumped without communication to an operator or special handling? This is particularly crucial if fourth generation languages are in the hands of users.
- Are all segmented library based changes made though job streams so that recovery of the system SL is reliable?

SOFTWARE DESIGN CONSIDERATIONS:

Consideration should be given to recovery when software strategies are developed. This is not to say that anything is taboo, only that the full impact be realized, and steps taken to prepare for disaster.

For example, if the system segmented library is used, the commands to place the application in the library should be in a clearly documented, obviously located job stream so that the addition of the routines can be done by anyone. Consider a group SLJOBS.SYS. If specific peripherals are accessed, identify them by class name, so that the recovery system does not have to exactly match the regular system, and also program the option of what to do if the device is not the one identified.

The accounting structure is critical to recovery - MPE V/E will build the structure as files are restored - but what of the user defined commands (UDCs)? We continue to recommend the use of the account building (BULDACCT) routines - and apparently others do as well, since the MPE V/E versions are available.

PEOPLE:

No matter how automated, or how labor intensive the business it will, eventually, resolve itself to people. To a greater or lesser extent, your users are dependent upon your skills and forethought to provide the computer tool to them. You, the significant few, provide, support and develop computer resources.

ORCHESTRATING AND PROTECTING THE PEOPLE -

We place significant emphasis upon four aspects of the people and their organization:

1) Have enough people -- identify in advance the people outside of the 'significant few' who have skills, interest, and availability to help through the days and weeks of a disaster. These may be purchasing, vendors, HP staff, other administrative types, consultants, etc. Recovery services should provide staff as well as the recovery hardware.
2) Define the roles, interrelationships, and training so that these people can all work together. Clear procedures are essential since some staff members may not be available.
3) Provide for the personal and family needs of these people so that if the same disaster disrupts their home and the business they are quickly available to the business. This means that key insurance, construction, legal, financial resources be made available to them.
4) Write it all down, so no one is uninformed, and provide the plan to all involved for review at each annual or more frequent test, as well as when it is initially put into service.

COMMUNICATIONS -

Of the Human sort:

During a disaster is not the time to establish who speaks for the company to outside parties. Nor is it the time to try to handle all the internal communication with users. Establish people in advance to fill such roles, and then direct all inquiries to them during the disaster.

Of the data sort:

Whereas in normal operation it is common that all users have access at any time to the system, this might not be true during a disaster. If multiplexor/modems are a key part of the network, it may be economically best to time slice a single communications link during a disaster, with the East Coast connected in the morning, the West Coast in the afternoon, or two hour rotations, etc. People are able to adjust for a reasonable length of time in a disaster.

SUPPLIES

Supply bottlenecks can be audited with every routine order, and the supplier contacted to identify and perhaps work out delays. The delays which cannot be avoided point to supplies which should be stored offsite in appropriate quantity.

Offsite storage of supplies may be possible with the supplier, which also provides inventory rotation. Movers and other storage companies often provide such service. Data storage firms are routinely providing secure storage of supplies as well as tapes.

A consideration in forms design and programming might be to provide a method of printing on blank paper particular information normally provided by the form, such as your company name and address on invoices. This would permit the emergency use of blank paper if required.

Summary:

In all of the areas noted, we can give many examples and options. The intent of this presentation is to spark interest in:

1) Evaluating the value of the processing for which you are responsible

2) Reviewing the advantages - protecting the business, the market and stock positions, management from legal actions - improvements in productivity, system performance, and staff capabilities.

3) Identifying and supporting the people who are the key to a successful recovery.

You have created a data processing resource to your company which is a valuable resource, and the effort expended deserves the protection provided by effective, well thought out contingency plans. The profitability and, perhaps, the survival of the company depend upon it.

Other reading:

(1) Heidner, Dennis, "Disaster Planning and Recovery", Proceedings 1984 International Meeting HP3000 IUG, Anaheim, California,February 26-March 2, 1984
(2) Savaiano, Richard A. "Disaster Recovery - Planning for the Unplanned", Proceedings 1984 International Meeting HP3000 IUG, Anaheim, California,February 26-March 2, 1984
(3) Lord, Kenniston W., Jr., The Data Center Disaster Consultant,Prentice Hall,Englewood Cliffs,New Jersey
(4) Disaster Recovery Planning Tools,UP TIME, Sacramento, California, 916-454-4171

About the author:

James A. Depp joined California On-Line Computer Services in 1984 after 13 years as a manager at Procter & Gamble. He is directly involved in the development and expansion of UP TIME, the premier disaster recovery service for the HP3000, providing dedicated hotsite service, either mobile to the customer's location or at the static site.

A user of H-P computers since 1971, Mr. Depp justified and installed one of the early HP3000's in Procter & Gamble's manufacturing facilities, and managed the operations and software development for that site. He has worked with the HP3000 for six years.

Responsibilities at UP TIME include the design and contruction of the mobile recovery vehicle, development of recovery planning tools, development of customer's recovery strategies, and marketing and sales.

3059. Information and Humanity

E. R. Simmons, Ph.D.
300 West Fifth, Suite 935
Austin, Texas 78701

A few years ago, before the advent of the smallcar craze in the United States and when the auto makers were trying to satisfy the desire of the American public for bigger and more powerful cars, one enterprising advertising adgency came up with a real gem to describe the power of their particular car. "More horsepower than you will ever need" was the message they declared to a public willing to buy anything in an amount more than they could possibly need.

I though that approach rather ludicrous at the time. Now, as I observe the love affair with information that has resulted from the ability of the computer to store, manipulate and access data in an unbelievably short time, I am aware of how vulnerable human beings are to any excess that happens across their pathway.

Information gathering in the United States has become a disease reaching epidemic proportions. We are not alone. Throughout the Western World of industrializd capitalistic systems, the same disease prevails to the extent that the epidemic has become pandemic. We insist on gathering more information about every subject known to mankind. It matters not that the topic may not be worthy of consideration in the first place or that the information we dredge up is not worth the cost of printing required to make it available to the public.

Information gathering has become an end in itself. Some apparently feel you can never have too much of it. Let's gather it, stack it, sort it, compile it, collate it, order it, systematize it and display it in a variety of ways and it will certainly be useful to somebody ssometime somewhere. It is accorded the same status as pure research in the Sciences. One never knows when the results of pure research are going provide the key to solutions related to mankind's well-being.

The contention here is that it is possible to have more information than you could ever want, much less need. Journalists have coined a phrase, "in-depth" reporting, to give this phenomenon the mantle of legitimacy. Frankly, it is still "more horsepower than you will ever need".

Our frantic search for more and more information and our determination to make it public appears to be nothing but an effort to capitalize on the weakness of human beings. We are always wanting more of nearly anything than is good for us, or even needful.

It is this writer's contention that pandering to human frailty is not a noble endeavor. Most of us will line up and do whatever anyone else is doing because we have been conditioned to that kind of behavior. Witness the great run on the purchase of personal computers. It is the new way of keeping up with the Joneses. We seem to enjoy being a "nation of sheep". Sometimes, we fall into line when we don't know what the line is for. Somehow, we have got the idea that there is something inherently American about following the lead of others

On the other hand, we must acknowledge the legitimate information explosion of our time, the result of advanced research techniques and technical processing and record-keeping systems. This is important to us as individuals and as a nation. The The problem is that everyone wnats to get in on the act and foist off on the public any notion of information they may have.

Information, as defined by one dictionary, is communication of knowledge, or knowledge derived from study, experience or instruction. In other words, one is gleaning something new from reading, listening or observing. Most of the time what is passed off as information provides nothing new. Quite often, it is justified by saying it is being presented in a new and more meaningful way. There is some doubt in the mind of the writer that presenting meaningless material in a "meaningful" way makes the content any more valuable.

While the scientific wizards and technical experts continue their adventures in the realms of electro-mechanics and the transmission of data, I think it is incumbent upon those of us who are concerned with human behavior and mental hygiene to point out the hazards of this unhealthy emphasis on information gathering. While we are looking for more and better ways to collect, process and present data, we need to look at the human side of the issue. The problems encountered by human beings in relation to the information explosion may be classified in two groups: (1) Physiological (2) Psychological

The first group have to do with the sensory system of the human being and its capacity for handling stimulation. The second group have to do with human reactions to pressures imposed by the situation. Several points need to be addressed relative to each.

In order to understand what happens to the human being physiologically, it might be good to look at the physical side for some examples from which we can make observations about the ccharacteristics of an individual physiologically.

From the standpoint of learning and making use of information, the human is a sensory system or sensory being. It is through use of the sensory systems that one learns about the environment and how to deal with it. Physically speaking, all systems in the human body have limitations. They are inherent in the structure. For example, there is a limit to the strnegth an individual has available for accomplishing work. The strength is related to the general condition, or health of the body. This, in turn , is determined by genetic heritage, care of the body in terms of nutrition, exercise, rest and freedom from disease.

Obviously, a person who is suffering from a debilitating disease will not be able to perform feats of strength that might be within his power when his health is sound. Neither can a person perform feats of strength that require extra xertion when he is already weakened due to long periods of exertion. These statements are easily understood and accepted. What is not so easy to understand and accept is that human beings have limitations intellectually and physiologically when it comes to handling information.

The point it that human beings can be incapacitated by being exposed to too much information.

It has been well established that the average person in his youth can learn only so much about a given topic in a given period of time. This is the very essence of the concept of curriculum development. We know how many repititions are required for a child to learn to count. We know how long it takes an average child to learn the multiplication tables. We take these bits of information along with other knowledge about the principles of learning and plan curricula accordingly. When we become adults, we forget the limitations we have and ignore general principles of learning and expect to accomplish the impossible.

If we present too many items for learning, the flow of information given interferes with the learning process and inhibits learning. If we present too many unrelated items for learning in a compressed period of time, negative interference may keep any learning from taking place.

So it is with the handling of information. If we overload a person with too much information, his ability to absorb it, digest it, file it away for future reference and recall it when the occasion demands will be hampered.

Another physiological problem arises when we assume that all people learn equally well through the same channels. Some of us, for example, are auditory learners. We tend to remember a great deal of what we hear. Others are visual learners. We tend to remember better the things we read or see. In many cases, it seems those who are responsible for the education of adults in job situations make no allowances for these differences in human beings.

We need to keep in mind also that the human mind has limitations as to the amount of input it can handle with dispatch. It has to develop its own filing system and methods of retrieval. It has to relate information received in such ways as to make sense of the total storehouse and organize it so as to have a useful body of knowledge.

Not all of man's information handling responsibilities can be turned over to machines and electronic systems. In fact, there is grave concern on the part of some educators that "real" education is being neglected in order to teach computer literacy and usage. It is as though some hope to substitute the computer for the mind. Why learn how to handle numbers if the computer can handle them for you? You don't really need to learn about algorithms and functions because the time is coming when we will not be re- quired to think at all. Just get the facts, ma'm, and feed them into the computer. Presto, the answer is there. It will not be as simple as that.

The computer is a tool to be used by a thinking mind, not one that has been taught that it doesn't need to think.

The final point to be made about the physiological aspects of the human mind in relation to the barrage of information thrown its way has to do with confusion that results.

It is well known and well accepted that the human sensory system can be rendered inoperational by being deprived of stimulation.
can be rendered inoperational by being deprived of stimulation.
What is not so well known is that it can be rendered inoperational by being bombarded by too many stimuli. In the former case, many studies have been done showing that disorientation takes place in a person when he is placed in a vacuum where there is no light or sound and where he does not feel any pressure or sense of cold or heat.

The human being depends on sensory stimulation to guide him into activity and help him determine his course of action. When he has no feedback from his environment, he does not know what to do with his limbs. If your eyes don't see and your ears don't hear and you get no clues as to whether you are in a friendly or unfriendly atmosphere, your mind becomes confused and does not know what to do.

Conversely, a mind that is beset with too many stimuli becomes confused and searches for assistance. The mind apparently can handle this up to a point. It can ignore some of the stimuli, but if there are too many to ignore or they are too intense, the mind becomes disoriented and leaves the field, so to speak. Sometimes, however, even though it attempts to leave the field, it takes the impressions and continuing onslaught of stimuli with it and becomes totally confused and develops irrational responses to all stimuli. Then, we have on our hands a person who is emotionally and/or mentally unstable and must be cared for by someone else.

It should be obvious to any thinking person that the implications for mental health from the standpoint of physiological well-being are too great to ignore. That brings us to the psychological problems. These are what give us so much trouble individually in our day-to-day activities. It goes without saying that we have a great need for information for a number of reasons. We have decisions to make every day, in business and in personal matters. No one wants to make decisions without facts to support them. On the other hand, we can have too much information for our own good. There are several areas of cncern for the person who is interested in mental health. We will look at them in the following order:

(1) Fear of having too little information (2) Fear of expressing one's self creatively (3) Fear of falling behind in one's profession (4) Fear of being ill-informed.

First, the fear of having too little information. This seems to be somewhat of a paradox on the face of it. We have been talking about having too much information to deal with and now we are saying this causes us to fear having too little.

A closer look will show why this happens. The very fact that there is so much information available today can make us feel that we do not have enough. We can get to the point where we feel that if we wait one more day or read one more journal, we will know enough to act. Meanwhile, we become paralyzed into inaction. We become fearful about making a decision. We never have enough information. Something is bound to be revealed tomorrow that makes what we now know obsolete. This can be dangerous to an individual in a business for obvious reasons. One is expected to stay up with what is going on and one is expected to make decisions readily and rapidly in business situations. To be rendered unable to make decisions is the worst thing that can happen to a person who is in a decision- making position or just has to make decisions concerning his own job from day to day.

Second, it can interfere with one's creative activity. We can become fearful of introducing new ideas simply because we are afraid what we have to say would not be consistent with today's level of knowledge about the subject. People who are in a high tech field are there usually because they are creative people. They are able to make contributions to a business other than just filling a job and doing what they are told. When they lose this ability, or willingness to explore new ideas out of far they will be ridiculed, they are not worth as much to their company. This, of course, effectively limits their usefulness and weakens their position with the company as far as advancement is concerned.

Third, the plethora of information that besieges us today can cause an employee to become so fearful that he will not be able to keep up in his profession that he may become constrained to try something different, giving up all of his training and leaving a job he may be doing very well, simply because he feels he is falling behind others with whom he works. He may not be able to see that he is not falling behind, that all are bothered by the same feelings. In such cases, it doesn't matter that he isn't falling behind. What seems to matter is the way he feels about it. And human beings quite often sell themselves short.

Finally, a person may come to feel so ill-informed that he becomes uncomfortable in any situation that involves other people, whether it is in his business life or in social situations. This does not have to do with thinking he is wrong about certain things, necessarily; but rather that he simply does not know enough. It may cause him to spend an inordinate amount of time trying to be a well-informed person, even to the detriment of his personal, social and business life. The time that he should be spending interacting with others is spent in isolation trying to overcome what he perceives to be his shortcomings.

Hopefully, these words of caution to those caught in the information deluge will suffice to make them take a realistic look at how they stand. If changes in thinking seem to be in order, it is hoped that readers will take corrective action.

3060. Response Time: Speeding Up The Man-Machine Interface

Tony Engberg
R&D Section Manager
ITG Performance Technology Center
Hewlett Packard Company
Cupertino, California

I. Introduction

Late in the fall of 1983, a small group of engineers was gathered together at HP Laboratories for the express purpose of characterizing the utilization and performance profiles of HP 3000 systems throughout the world. Using an automated data collection package (about which you will be hearing more in the coming months), and a set of survey forms designed to collect information of a qualitative nature , this team amassed a sizable cllection of data which profiles, quite clearly, the uses which our customers make of our systems, and the manner in which our systems reponds to those uses. The results of the study have been far-reaching; they have affected the design of future HP systems, they have provided the basis for new tools for customers and HP engineers, and they have determined the directions being taken by many of our product lines. Most importantly, however, they have shown what is, and is not, of importance to our costomers when purchasing and using our systems and system components. One of the study's findings, for example was that, when it comes to assessing system performance, the metric most users consider important is response time. This paper will discuss response time in a manner which will allow the reader to fully understand the many factors which have an impact upon it, and which will guide those concerned with improving response time characteristics in applying their efforts most effectively.

II. Definitions

Although it is often possible to carry on a discussion of some concept without providing a rigid definition of the subject at hand, this is not one of those times. Actually, this is and is not one of those times, but you'll have to bear with me in order to see what I mean. So, before proceeding on, answer the following question:

"What is the definition of response time?"

Now, consider the following answers to the proceeding question:

1. Resonse time is the time it takes the system to respond to a user's request.

2. Response time is a measured interval which begins when an interactive user transmits a record to the system (by striking the carriage return key, or ENTER key, or what-have-you), and which ends

when the system sends the first character back in response to the user.

3. Response time is a measured interval which begins when an interactive user transmits a record to the system (by striking the carriage return key, or ENTER key, or what-have-you), and which ends when the system is again ready to receive more data from the user. These three definitions represent those most frequently given by designers, programmers, and end-users whom I have queried about response time. Let's examine them in an effort to distill a single definition of this important metric.

"Response time is the time it takes the system to respond to a user's request." While intuitively appealing, this definition leaves much to desired. It is ambiguous. What are the bounds which define the "time" it takes a system to respond? What is a "user request"? What constitutes a response from the system?

I can here the grumbling already. Most people who provide this definition assume that there is a mutual understanding of the bounds upon the time interval, and that a "user request" is always understood to be an interactive transaction, and that system responseis always clearly delineated. Believe me, this is not the case. The second and third definitions provide but two of the myriad ways in which the response time interval can be, and is, quantified. A "user request", in many dicussions, includes the submission of a batch job to the system. The second and third definitions also indicate two differing views of what constitutes a response from the system (i.c., first character or ability to continue). Our ability to improve response time will depend heavily upon our ability to unambiguously define that concept in a matter which will permit measurement. If we cannot measure response time we cannot discuss optimization, except in the most qualitative sense. Let's move on, and examine the second definition.

"Response time is a measured interval which begins when an interactive user transmits a record to the system (by striking the carriage return key, or ENTER key, or what-have-you), and which ends when the system sends the first character back in response to the user." Is this definition, known as the "transmit-to-first-response" definition for response time, better than the first? It certainly removes much of the ambiguity. The time interval is clearly defined; it could be measured with a stopwatch. "User request" has been limited to an interactive transaction. The transmission of a character back from the system to the terminal has been selected as the delimiter for system response. What more could one ask?

Quite a bit, I'm afraid. Consider the system designer who has taken into account the qualitative nature of response time by incorporating a message intended to sooth the impatient user into the processing stream. This is a common tactic. The computer onboard the fictitious starship Enterprise used it ("Working..."), the HP2647 games tape used it ("CRUNCH*CRUNCH*CRUNCH"), and many user programs use it. Should we consider system responsiveness to be delimited by the printing of such a message? Clearly not.

"Wait a minute", I hear you grumbling again, "that's easily handled. Simply use the first character following completion of processing as the delimiter." Good suggestion, but this still leaves a problem. What if the system has insufficient resources (CPU, memory, I/O, or others) to display the entire response quickly. Think of the times you have seen a screen or so of data come up, then waited as the system went off and did other thingsbefore finishing with your output. Think of the stories you've heard about cursors pausing in mid-screen, then moving on. It is reasonable to measure the response time interval up to only the display of the first character in such cases? I doubt if most entry clerks would think so. Let's try the third definition and see if we can find a better means of describing response time.

"Response time is a measured interval which begins when an interactive user transmits a rcord to the system (by striking the carriage return key, or ENTER key, or what-have-you), and which ends when the system is ready to receive more data from the user." This is the "transmit-to-read" definition of response time, and it was formulated to overcome many of the objections raised to the preceeding two definitions. It is less nebulous in phrasing than the first definition, and it gets around the "intermediate output" and "delayed response" Shortcomings of the second definition. So, at last, we have a usable, working definition of response time.

Well, not quite. This last definition has some pitfalls of its own. To begin with, it is very much dependent upon variables introduced by the designer and programmer, and by the terminal equipment used. Consider, for example, the implementaion of an application using fairly verbose data entry screens. Should the time required to paint a screen be included in the sytem response time? Many users would say yes, while many others (interested only in how fast the system actually took to handle the last request) would say no. For purposes of comparison it is often preferable to measure only the time spent by the system, excluding the terminal I/O time, processing the request.

Now, your probably wondering what this continual refutation of definitions is all about, and wishing that I would just come out and tell you what response time is. Well, I can't. I'll pause while the grumbling subsides to a dull roar. You see, response time is really a qualitative concept which can only be quantified in the light of a specific application or environment. That is to say, it's definition must vary with the application, user needs, and system. At its root, response time is a perception (hence the tricks mentioned above for stalling the user). Only you can decide what sould be measured and called "response time" in your environment. Let me give you some further examples of the diversity for which you must account.

1. You are running a shop in which data entry clerks are using the system to key in short records which the sytem then validates before allowing further entry. Response time is characterized as the amount of time the entry clerk must wait before being able to proceed with the next record, that is, transmit-to-read.

2. Your system has an on-line document look-up system. Those needing to see a particular work enter a few key words, hit carrige return, and wait until the document is found and displayed before them. They peruse the located text, determine its applicability to their task, then print it if they need a copy. Response time is measured from the point at which the carrige return key is struck until the system starts to print out the document (i.e.,transmit-to-first-response).

3. You tag shipments with special weight and handling stickers produced by the system. A crate rolls up to an entry station, its contents is read in by a clerk who runs a wand over a label on the crate, it is weighed, and special handling instructions are keyed into the system. The system takes these inputs and generates a tag which is taken from a local printer and applied to the crate immediately. Response time is measured as the interval extending from transmission of the descriptive data to completion of the printing of the lable.

It must be pointed out at this juncture that, while you are defining response time in your environment, you must consider not only what interval to measure, but what tolerance you can allow (how much variance around the average, if you will). Why is this important? Consider the data entry shop described above in the first example. Let's say that the average response time is 1.5 seconds. We got this average by measuring ten separate transactions; the times recorded ranged from .1 seconds out to 4 seconds. Although the average may be quite adequate for the type of work being done, the variance may be totally unacceptable. Data entry clerks are slowed, in many cases, when they can not establish a rhythm. Reporting response time as an average is all well and good, but pay attention to the other details of import. In short, realize that you are dealing with perceptions, and decide what, and how, to measure accordingly. Now, before proceeding, think about your shop, your applications, and work out the points you would choose to delimit response time if you had to measure it. Also, before proceeding, analyze the amount of variability which your applications can tolerate in response time. This last challenge can be met, in a somewhat rough way, by simply estimating what an acceptable response time would be, and what an unacceptable deviation from that response time would be. Once you've worked this out, proceed.

III. Components

No matter how you have chosen to define response time, the following formula can be employed in any effort to optimize it, since the parameters within the formula can be defined to include only those items which fall within your response time definition:

RT = the time that a request spends in the system,

therefore,

$$RT = Tc + Ti/o + Tm + Tl - Tovl$$

where:

RT = response time
Tc = time spent at the CPU
Ti/o = time spent in the I/O system
Tm = time spent waiting on memory management
Tl = time spent waiting for lock and latches
Tovl = time spent overlapping functions (such as CPU & I/O)

Note that the above times can be broken down even further, as folows:

```
Tc = Tc(q) + Tc(s)
               Tc(q) = time spent waiting for the CPU
               Tc(s) = time spent being serviced by the CPU
Ti/o = Ti/o(q) + Ti/o(s)
               Ti/o(q) = time spent waiting for the I/O system
               Ti/o(s) = time spent being serviced by the I/O system
Tm = Tm(q)
               Tm(q) = time spent waiting for memory
Tl = Tl(q)
               Tl(q) = time spent waiting for locks and latches
```

Once response time has been broken down into its component parts, as above, analysis and optimization become easier. Observe that each component is affected by system loading. Thus, the time spent at a particular terminal waiting for the CPU will be a function of what other processes in the system are contending for the CPU, what their resource consumption profiles look like (e.g., how much CPU do they consume at a time), and their priorities relative to that of the process corresponding to the terminal in question. It is not my intent here to provide a full description of all the items which affect each component, but merely to describe those which have the greatest impact on response time and to suggest those things you ought to be thinking about when considering ways to minimize this impact.

IV. Tc(q)

CPU time can, as we have seen, be described in terms of two components: queueing time, Tc(q), and serving time, Tc(s). The time spent queued for the CPU will depend upon a number of things, most important of which are the priority of the process waiting for the CPU, the lenth of the queue of processes of equal or higher priority, the service times attached to those processes ahead of you in the queue, and the rate at which higher priority processes are arriving at the CPU. Let's look at each of these in turn.

First, the priority of your process will determine where in the dispatch queue (the line of processes waiting for the CPU) your process "gets in line". An excellent discussion of how your priority is set, and the impacts of other processes upon your ability to compete for the CPU, is provided in a paper on this subject being delivered at these same proceedings by Dave Beasley. Let it suffice here to simply realize that:

1. In order to reduce queueing delay, you should be at the highest possible priority. As you compete with process of equal or or higher priority, your time spent in the dispatch queue will lengthen, and so will your response time.

2. As you compete with processes of equal or higher priority, your response time will, most likely, increase in variance. This increase is due to the lack of similarity in resource consumption profiles. In short, the more processes, the more likely there is to be variability in the amount of CPU time each demands.

If response time for a particular application is critical, that application should be favored when entered in the dispatch queue. This can be done in several ways, including manipulation of the queueing structure (via the TUNE command) or programatically (through such mechanisms as the GETPRIORITY intrinsic). The latter will also reduce the variance in response time in most cases. REMEMBER, however, that FAVORING ONE PROCESS (or process set) WILL IMPACT OTHERS. The MPE operating system has been designed to treat all interactive activity equally, and to permit the user to decide how much to favor CPU-bound processes (again, via the TUNE command). In an environment in which the object is to optimize response time for all processes, and in which multiple applications are running concurrently, tricks such as priority adjustments are not the answer.

The second item to be considered was the length of the queue of process of equal or higher priority waiting for the CPU. We've already discussed one way to shorten the line in front of you, and mentioned the problems which this can cause. You should realize, however, that applications which don't need priority treatment can interfere with those that do. For example, batch jobs which are sharing a resource with interactive process can (and often do) reach prioritylevels which allow them to line up high in the dispatch queue. Again, I refer you to Dave Beasley's paper for a discussion of this phenomenon. I recommend strongly that, in an environment in which response time is crucial, you fully evaluate the work which is being done on the system and the manner in which processes interact.

The next item, service times of processes ahead of you in the dispatch queue, is one over which you have little control. Techniques and tools exist for measuring these times, but these are not currently available to the user community. The best you can hope for is that each process ahead of you has been coded by someone who has optimized for Tc(s) (see below), so that they get in and out as quickly, and efficiently, as possible.

Finally, the arrival rates of processes at a higher priority than yours to the dispatch queue will directly impact the value of Tc(q) for your process. In most cases this should not be a problem, provided that you understand the use of the TUNE command, and set your queue size, queue overlap, and the average short transaction time intelligently (and don't have someone else calling GETPRIORITY).

V. Tc(s)

Once you've done all you can to reduce the amount of time you spend waiting in the dispatch queue, the next item of interest is the amount of time you spend actually using the CPU. The key concern here is pathlength; you don't want to waste any time when you are in CPU. Remember, the Tc(q) of other persons on the system is dependent upon how long you hold this resource.

How do you go about reducing Tc(s)? The answer is, of course, to code efficiently from the start. The answer is, also, to isolate those portions of your code in which you spend the most time, and to make these sections as efficient as possible. The best way to do this is through the use of a tool such as APS/3000 (often called "SAMPLER"), which will provide execution traces of your programs which show the fequency of time spent executing within specific ranges of instructions. This will enable you to isolate the most heavily used sections of your code and optimize the algorithms which you use in these sections.

VI. Ti/o(q)

Let's move out of the CPU now, and over to the I/O system. The amount of time you spent queued for I/O depends upon a number of things, most important of which are the priority of your I/O request (for disc I/O), the length of the queue in which you are waiting (which, in turn, depends upon the arrival rate to the queue), and the Ti/o(s)'s of the requests ahead of you. For the purposes of this discussion I will use disc as the I/O device for which you are waiting. Disc is the most complex case (since it is shared and prioritized); other devices can be understood fairly quickly once discs have been grasped.

Your place in the queue for disc I/O depends upon the priority at which your process is running. This really is a priority-based system, and you have to be concerned about where you are running. Section IV has already covered this area.

The length of the queue in which you are waiting will depend upon the number of I/O requests which have been directed to the device upon which you are queue, and the time required to service those requests. This means two things. First, it is imperative that you spread your I/O across multiple drives, since the more requests you direct to a single disc the longer the queue on that drive will become. Second, you should attempt to localize your I/O (reducing seek time) as much as possible on a drive-by-drive basis. How do you do all this? You ballance your discs by splitting files which see heavy access across multiple drives (particularly file which tend to be accessed in sets). You localize your I/O by placing the most heavily accessed files in close proximity upon your drives (preferably centered on the disc).

I realize that this latter activity is an extremely painful one to undertake, since it requires successive partial RESTORE's to specific devices. It is, nonetheless, one means of reducing seek time. The service time of requests ahead of you in the queue is, as with Tc(s), something over which you have little control. The next section will discuss this topic.

VII. Ti/o(s)

How long you spend receiving service from the I/O system is a function of the speed of the device which you are utilizing, the size of the request you are making, the efficiency of your request, and whether (in the case of disc) your request can be resolved in cache. The speed of the device is a given; it can be reduced only by substituting a faster device (although the efficiency of your I/O request is related to the device's physical characteristics). Those with critical response time concerns might review their hardware configuration in this area, since such items as transmission speeds can have a very strong effect upon response time (depending upon how you've defined it).

The size of a given I/O request within a transaction, and the efficiency
of that request, will have a definite impact upon service time. Specific disc drives, for example, can retrieve and transmit data much more efficiently within given size limitations. The file system (which most end-users would consider part of the I/O chain) will respond much more quickly to certain types of I/O requests than others(e.g., unblocked, NOBUF transfers vs. blocked, buffered). The trick here is to examine the I/O which you are doing, and don't assume that defaults are always the best way to go. Again, try to spread your access across drives, and localize the targets of your accesses.

Resolution of read requests in memory-based cache obviously shortens the I/O service time considerably. Realize, however, that write request must, of necessity, cause cache domains to be flushed. One way, in some cases, to reduce time spent in the I/O system is to localize your write-intensive files on specific drives, and to turn cache off on those drives. This requires experiment, but it is worth looking into if you have a sufficient number of discs to to allow it (remembering the impact of long queue lengths for any device). A side benifit may be a reduction in the amount of CPU resource expended to manage the cache (reducing, possibly, Tc(q).

VIII. Tm(q)

The amount of time you spend queue for memory will depend on your priority (of course) and the amount of time required to find memory space for your process.

When the dispatcher selects your process to run (based upon your priority) a check is made to see that you have everything required by your process to run in memory (e.g., stack and code segment). If you do not, your process must wait while the memory manager attempts to make space available. Thus, your priority determines when the system begins to look for space for you. Once the search has begun, however, other factors come into play.

The memory manager will read down a list which contains those segments which you need in order to execute. Each time it finds one which is not in memory it will check the size of the absent segment, and attempt to find an available slot in memory into which it can fit that segment. If no such place exists, the memory manager will begin a trip through memory aimed at making such space available. The mechanics of this trip are not important here (although they are worth knowing). What is of import is the things which extend the length of time this trip takes, specifically:

1. Segment size. If the segment which the system is trying to bring in is larger than the segments which are currently in memory, it will take longer to find space and, in all probability, will result in multiple processes losing segments (meaning more memory manager overhead later).

2. Memory size. The less memory, the longer it will take the memory manager (on a busy system) to come up with an available chunk of memory. Also the less memory, the more likely it is that that chunk of "available" memory actually belonged to someone who will need it again shortly.

The best answer, of course, is to avoid the memory manager altogether. This is not an easy trick, but you can enhance your chances to avoid being swapped out by observing the old recommendations that suggest:

1. Stay in a segment for as long as possible. When you leave, stay out for as long as possible.

2. Share code where possible (the argument being that the more processes sharing a piece of code, the more likely it is to be referenced and, therefore, maintained in memory).

IX. Tl(q)

The time spent waiting around for locks and latches can be significant, and the avoidance of this delay is often much easier than one would think. First let me point out that you should NEVER ignore a lock unless you are certain of the consequences. Given that caveat, let me go on to recommend the following:

1. Don't use locks unless you need them. Don't, for example, lock around reads unless there are data integrity problems caused by concurrent writes.

2. Don't hold locks of any kind longer than absolutely necessary. Batch jobs which hold locks required by interactive processes will cause all kinds of havoc.

3. Be aware of the fact that many operations cause serialization on lock or latches over which you have no control (other than to avoid whatever the operation was that drove you to hold that lock or latch). Examples include purging items in the system directory (requires a system lock, called a System Internal Resource), data base accesses (serialize around the Data Base Controll Block), and some sub system lookups (which lock, but don't tell you).

X. Tovl

There is one parameter in the response time equation which is subtracted out rather than added in: overlap time. Anytime that you can overlap I/O operations with other times, by utilizing NOWAIT I/O, you are reducing your overall response time (assuming that you take the care to implement this efficiently). I advise caution in doing this; time should be spent studying the proper use of NOWAIT I/O and the trouble which you can get yourself into if it is mishandled. Used properly, however, it is an effective weapon in reducing response time.

XI. Summary

The purpose of this paper has been to introduce the idea of response time as an application and environment dependent cocept, whose definition must take place within bounds imposed by the end user. Further, I have attempted here to provide a description of response time in terms of the components.

It would take a book to fully describe the mechanisms which go into determining the queueing delays and service times encountered by any process at each of the components delineated. I would encourage those who are serious about optimizing response time on their system to consider carefully the paradigm provided here, and to then expand their understanding of the various components through such vehicles as courses (particularly on MPE system internals), manuals (especially the MPE System Tables manual), papers and articles, and books. I will be more than happy to correspond with those interested in pursuing this subject in greater depth (since we have only scratched the surface here).

3061. AUDITABILITY
or
WHAT'S A NICE BYTE LIKE YOU DOING IN A BASE LIKE THIS

Robert A. Karlin
7628 Van Noord Ave.
N. Hollywood, California 91605

Friday. 3:45pm. The sun shines through your office window filled with the promise of a grand and glorious weekend. Tomorrow's picnic vies with the tickets to Sunday's game in their efforts to banish the code in front of you from your mind. As the idea of packing it in early begins to slowly insinuate itself into your consciousness, you hear an embarrassed cough, followed by a knock on your door.

"Sir?".

"Yes?".

"There seems to be a slight problem with GL.".

"Yes?".

"Well, the weekly doesn't balance to figures that accounting gave us.".

"How much are we out?".

"About fifteen dollars and some small change.".

You breathe a sigh of relief. Visions of thunderclouds recede from your imagination. Glancing at the clock, you muse that at least you will have something to keep you busy until it is time to check out.

* * *

9:30pm. You know, it wouldn't have been this bad if the operator had warned the users that you were taking the system down at four to research the problem. As it was, the system was in the middle of a chained delete, and you think that the pointers are totally gone. Well, this is what they pay you for.

* * *

11:30pm. The dump of the data base should be about finished. You never realized that 400 lines per minute was so slow. You contemplate making a fourth pot of coffee, and then turn back to the list of daily transactions, checking them off, one by one from last weeks Open Item to this weeks Open Item.

* * *

3:15am. The numbers in front of you blur as you attempt to reconcile the figures in front of you with what you know should be on last weeks backup. Both errors that you have found so far have increased the total figure that you are out by about three thousand dollars. The couch in the hall begins to look very good. They really don't pay you that well, do they?

* * *

Saturday 2:30pm. The tape drives have got to go. Really, is it too much to expect that out of three backups, one would be good? Somebody should have noticed last week that the tape that was restored was missing the last part of the account file.

* * *

Saturday 4:30pm. The operators have got to go. It wouldn't have been as bad as all that, but you've been running for five hours using the wrong input tapes. Can't those guys tell a 1 from a 7?

* * *

Sunday 5:00pm. The game didn't sound too interesting on the radio, maybe it's best you missed it. The data base is finally correct (thanks to some fiddling with DISKED2) and all you want to do is go home and relax. You finish your instructions to the operator to back up the system tonight when he gets in and leave for home.

* * *

Monday 1:00am. Rrring...Rrring...Click "Hello? Sir? We're getting WCS parity errors and the CE thinks we've bombed our data pack. Which file should we reload from?".

* * *

The above represents the classic Data Processing nightmare. Anyone who has been in the field of Data Processing for more than a few years has seen similar occurrences. Our hardware is not perfect, and our software will have bugs. No large system will be completely free of them. Our task as DP professionals is to minimize the effect of bugs and crashes. This includes having our hardware PMed regularly, buying good tapes and certifying them after they have been used for a while, adequately testing our programs before implementation, and insuring that the system is auditable. The last of these is the scope of this paper.

AUDITABILITY

Auditability. From the word AUDIT, from the latin 'audire' meaning 'to hear'. In order for you to 'hear' your system, it must say something. In order for the things your system says to be useful to you, you must listen to it. Auditability is comprised of these two halves. The first half consists of the techniques of coding and design that we will discuss in this paper. The second half consists of the departmental procedures (including adequate staffing of a production control department, user and operator training, etc) to balance and check the output of the first half. Without the second half, the first is worthless. One particular system that I am aware of was doomed to failure because the weekly reports sat on a clerk's desk for six weeks before an attempt was made to balance them. By that time, the system was so out of balance that it was impossible to trace the reasons.

There are three basic areas that we will cover in this paper. The first of these is DATA INTEGRITY. How do you design early warning signs to detect when your data has been corrupted. The second is the AUDIT TRAIL. How do you find when and where your data has become corrupt, in order to isolate the error and prevent the problem from reoccurring. The third is PROBLEM RESOLUTION. When you discover where the error is, how do you fix it while still maintaining auditability.

DATA INTEGRITY

Insuring Data Integrity can be difficult. Aside from your massive crashes, you still must deal with program and data entry errors. If your system is a mailing list for a mass mailing house, dropped records are probably not one of the great disasters of modern time. On the other hand, if you are a major bank, dropped records can be the company's death bell. Most of the techniques we will discuss involve some overhead. The method or methods that you choose must be comensurate with the risk involved to your company by the failure of your system. Some of the techniques build on others that we discuss, but there is a way of implementing each independently.

THE FLAG FILE

The first techniques we will discuss concern system failure. Many data bases are large enough to preclude restoring the data base in the event of system failure, if at all possible. One very simple technique of assuring the integrity of your data base is the Flag File. In its simplest form, it consists of an unblocked file with one record for every possible terminal on your system. Each record is set to true if the user working on that terminal is in the process of updating your data base. If, after a system crash, you find that no records are set to true, you can bring up your application with no worries. If you find that one or more flag records are set to true, you can check with the user on that terminal to find out what he has been doing. Interrogating the data base with QUERY, INFORM, or an application maintenance tool, should allow you to determine the actual state of the transaction. All printout should then be kept to document the state of the data base after the crash. Weeks later, it could be extremely important to note that a particular problem did or did not arise as the result of a system crash.

To establish your flag file, build a 1024 record unblocked MPE file. The record length is not important, but the best length is 256 or less. The index into this file is your Terminal logical device number (LDEV). When you are about to add to or update your data base (Image, Ksam, Flat file, etc) you issue a write direct to the file setting the first word of your record to true (or - 1). Then you must insure that the flag record has been written to disk (by issuing an FCONTROL 2 to flush the buffer). Now you are ready to do your updates. After your transaction is complete, issue another write direct to your file, setting the first word of your record to false (or 0). A program to print out the LDEV of every record that has its first word set to true is trivial.

THE SCRATCH FILE

Another technique is the Scratch File. The scratch file is used to complete any outstanding transactions when the system is brought back up after a failure. The technique makes use of the capabilities of MPE message files. When processing the data for an update to a series of records in your data base, instead of processing the updates at that time, they are written to a message file and then read back into the program and processed. If the system crashes during the update, the unfinished portion of the update is still stored safely on disk.

The scratch file is a permanent file whose record length is equal to the largest update record of the system, whose filesize is greater than the largest number of separate records in any one update, and whose name contains your terminal id, or some other unique identification. After the updates have been completely written to the message file, we write a 'bracket' record that will be identifiable as the last record in the file and tell MPE to update the end of file pointer (by issuing an FCONTROL 6 (write eof)). If we are using a flag file we now set the critical flag, and the message file is now read and processed. Before each read, we tell MPE that we wish to read the current record from the message file without destroying it (by issuing an FCONTROL 47). We then process the update. Then we issue another read, this time without preceding it with the FCONTROL. This will 'pop' it off the message file. We continue until we have reached the last record of the file. If the system fails during the process, the update can be finished when the system is brought back up. During the system design, you must decide whether a program should be written to finish applying the update, or the application should finish applying the update on initialization. Since the records are processed before being destroyed, the only problem to be coded for is duplication of the record that was being processed at the time of the failure.

RECORD COUNTS

The most important rule of auditability is: Count Everything. All transactions that affect the data base should be tallied somewhere. The easiest early warning sign to recognize is record counts that do not match. The number of records in yesterday's data base plus the number of records added today, minus the number of records deleted today must equal the number of records in today's data base. If not, we have a problem. Count everything. In fact, all transactions should be counted twice. We will discuss this point in greater depth when we discuss audit trails. For now, we need a place to store all of these counts we are generating. The best place to store them is in our flag record. Because of the way HP treats unblocked files, we already have 255 spare bytes per flag record that HP will reserve for us whether we use them or not. So we may as well store our counts here. Every transaction type should have its own counter. Every record that is written or read should also have its own counter.

There are many side benefits to this scheme. Primarily, you can use this to determine if you have too many terminals attached to your system. If you discover that certain terminals are used maybe once a month, you can certainly ask your user to justify the expense of tying up those terminals. You can also see at a glance what transactions are being heavily used, and which ones are lightly used. When response time problems arise, this information can be used to determine which modules should be looked at in your drive for more efficient use of resources. Even if you do not code the software to interpret the counts, design your system to write them out. It is usually three to five lines of code per program to add these routines. This is miniscule when coding the original program, but is a tremendous task to retrofit counts into a system of seventy or eighty separate modules.

THE HASH TOTAL

Another technique that can be widely used is the hash total. The hash total is so called because, like corn beef hash, it doesn't matter what goes into it, and what comes out looks nothing like what went in. The hash total is a simple way of verifying that what went into one's data base is what is still there. The hash total involves either taking the whole record, or just the more important fields, redefining them as an integer, then adding them all up into a hash total. If you hash every add or update to your data base, and keep track of the result, you can verify the total by reading your base sequentially and checking the tally against your total. This technique is very useful for spotting unauthorized updates to the data base (such as with QUERY). The best place to keep your hash total is in the data base itself, in a data set that has been created for that purpose. This will certify that the hash totals that are to match this particular data base are stored with it. Every program that sequentially reads a file or set that has a hash total associated with it should hash the file and print the result. This can be easily checked against the data base to verify the integrity of that file.

THE AUDIT TRAIL

When I first entered Data Processing, I heard the term Audit Trail and thought of a dusty dirt pathway through the California scrub brush, where herds of cattle were driven to market. As I became aware of what an audit trail was, and had to live with interpreting them, I realized that my first impression had not been far from wrong. An audit trail has many of the same characteristics of a cattle trail. It should be easy to follow, and not twist and turn at every step. It should be plainly marked, and should not intersect other trails, or fork into two undifferentiable roads. And it should have clearly marked termini.

There are many ways of marking an audit trail. You can set signposts, and you can set toll gates. You can set guards to allow only those with the correct passwords to pass. You can even create detailed maps of the terrain. But you must at least have some way of knowing all who have passed.

The entire purpose for an audit trail is to provide a means for verifying the source of all data in your data base. Your audit trail must provide you with an easy means for tracking down any datum that you may consider spurious to determine how it got into your data base. It must also provide a means for verifying the exact sequence in which events happened.

THE AUDIT LISTING

The most important consideration in designing your audit trail is that absolutely every change to your data base must be recorded at the detail level. Image logging is an alternative to writing your own log file, but, if you elect to use it, you must write a transaction formatting program to interpret it. You must also include additional information, using either DBMEMO or including extra fields on your update list, since Image only logs the record number for deletes, and the affected fields for update. In researching a problem recently using an image log tape, it was disconcerting to try and trace a series of transactions that consisted of only a bunch of records, somewhere, that had a field called DELETE-FLAG updated to a 'Y'.

If you elect to use your own audit trail, you must be consistant. First, and foremost, you must disable QUERY and any other program that does not write to your audit file from updating your data base. Second, no application fix can be allowed to be written that does not write to the audit trail. Third, and this applies if you use image logging as well, your audit trail must be able to be printed in at least two sequences; that is, in absolute time order, as the transactions happened, and in the sequence of either your detail trial balance, detail open item report, or other detail data base listing. And, of the two, the second sequence is the most important, since the first can be simulated by dumping the transactions in hex format to the printer. This report should also contain the same totals as the report it will be checked against. If your bottom lines do not balance, it should be a relatively easy task to add your audit report to your last open item, at the group level, to pinpoint the source of your out of balance condition.

SECURITY LOG

The security log actually has less to do with security than with auditability. The security log consists of sequential time stamped records created when a program or transaction starts or ends. The best way of implementing the log is through a separate subprogram, compiled into an account SL, that is called by every program that accesses the data base. The parameters passed vary with the application, but should at least include the program name and an action code. Large menu driven applications can write records to the security log at every menu step, or at the entry to a particular function. The log should be detailed enough to identify what transactions were accessing the data base at any particular time. The log can be used to track down problems that occur due to improper locking strategies or other timing problems. It also records the fact that a particular user actually did enter a particular program at a particular time.

PROGRAM VERSION CONTROL

One particularly nasty problem in tracking down problems is verifying the particular version of the program in production. This can be especially acute when the problem occurred weeks in the past. There are two techniques for establishing version control over your programs. First, you can store the date of the last modification and a version number within your program. This information should be printed in the heading of all reports, and can be stored in the security log and audit file during program initialization. This technique involves strict programming standards, and must be followed for every program change, however minor. Second, you can write a subroutine to open the program file itself, and return the creation date of the file. This should also be printed in your report headings and stored in the security log, and audit file. If you use this technique, be certain that all MPE restores done for your system use the OLDDATE parameter to prevent the destruction of the creation date of the file.

RECORD COUNTS

We have seen previously that record counts can be a superb tool for detecting data integrity problems. Record counts are also very necessary for a complete audit trail. Every program that alters the data base in any way should store the number of records affected in two separate places. First, we should store our counts in our flag record or in another cumulative place. This is an overall view of what this particular user or terminal is responsible for. Second, every program should log its record counts at termination to either the audit file in special count records, another sequential file, or the security log in the logoff record. Every batch program should, as well, count all input and output transactions. Counting records should become so completely ingrained in the mind of a programmer that he could not conceive of an input or output routine without adding to a counter. These counts can be invaluble in determining where things may have gone out of kilter and, more important, where they didn't go out of kilter, by verifying the proper sequence of steps, and certifying the correctness of the input data.

THE RACN NUMBER

RACN is an acronym for Run Activity Control Number. The purpose of the RACN number is to identify a particular version of a data base. The RACN is an integer that is incremented every time the data base is opened for update and every time it is closed after being opened for update. Its primary purpose is to identify the exact version of the data base used for a particular report, or as input to a particular batch update. The RACN should never be odd when the data base is closed. The RACN should be printed on every batch report and batch update. It should also be recorded at the end of the processing day either when backups are started, or when the log file or transaction file is closed, and entered both in a control log and on the tape label. The RACN serves to sequence batch reports in time order, allowing you to be certain that time dependent reports were run in the appropriate sequence. The RACN can be recorded in the security log to pinpoint the sequence of logons and logoffs accurately, and serves as a unique identifier for a particular session that can be used in your audit file or in the actual data base record.

PROBLEM RESOLUTION

The scope of this paper is not to teach you how to recover from an error in balancing or from a head crash. That would be another paper in itself. The purpose of this section is to present ideas on how to correct errors without destroying your audit trail. In general, all corrections must follow the techniques we have discussed so far. There is no excuse for bending the rules because this is a programmer correction and not a user transaction. If possible, the actual changes to your data base should be made by a production support person, or, if the shop is to small for that position, the manager of the Operations department. The programmers, if they are involved at all, should be on a strictly advisory nature.

QUERY ET AL

The most important rule while using QUERY or any other generic data base manipulation program to correct your data base is DON'T. QUERY is an easy method for correcting the out of balance conditions without regard to the internal integrity of your data base. More problems have been caused because of improper use of QUERY to fix a data base than any one factor that I have come across. MPE allows you to restrict QUERY access to a data base to read only using DBUTIL. Enabling this feature will save your shop from days of attempting to discover what went wrong with a program, when the culprit was a fumble fingered programmer in QUERY.

THE MAINTENANCE PROGRAM

"If I can't use QUERY, what can I use"? The specs for any project should include a program that reads, writes, and updates your data base on a set by set basis. This program should verify that all fields are logically correct as well as physically correct (i.e. that if your invoice header says that there are ten line items, there are ten line items). The program should also update the audit file during these updates. There should also be an option to create special journal type entries to your audit file, to allow you to bring your audit file into balance if necessary. These records should be easily identifiable on any report as additional records not related to the main user programs. The maintenance program probably should be the first program that you write in your system, and should be maintained carefully when the data base changes.

DOS AND DON'TS OF PROBLEM RESOLUTION

DON'T modify any record on your audit file. Always add an offsetting entry if you need to make an adjustment.

DON'T add to or update your data base without adding an entry to the audit file. If it is necessary that the numbers not be added to the bottom line of your audit file, add an offsetting entry.

DON'T use QUERY, or any other update program that does not add to your audit file, and does not check logical relationships.

DO record every step that you take, and file them in a convenient place.

DO keep a listing of 'before' and 'after' with your problem documentation.

DO keep a log of production problems, and their solutions, filed by problem type, to allow quicker problem resolution.

DON'T let programmers modify your production files. If they are the only ones who know how, have them write a procedure for Operations personnel to follow.

IN CONCLUSION

FRIDAY. 3:45pm. The sun shines through your office window. This weekend you are going to leave on time if it kills you. It has taken all week to recreate the work you had done over the weekend. It probably would have been easier, but the user had thrown away all of his past week's reports, and you had to compare the current open item with the data base on a line by line basis, and there were 450,000 lines. Well, it's over now, and the weekend looms ahead. But as you take your feet from the desk and push your listings into a neat pile, you hear a knock on the door...

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

George B. Scott
Eldec Corporation
16700 13th Ave. W.
Lynnwood, WA 98046-0100
206-743-8227

I. WHY

One might ask "Why use process handling?". Our reasons were (1) to eliminate the need for user knowledge of MPE syntax, (2) to facilitate management of a number of user terminals in a multi-CPU environment and (3) to optimize performance by sharing certain overhead and resources.

This paper presents the requirements (Section II) which led to the selection of process handling as an alternative, the environment/configuration (Section III) of the system, the process tree and function of each process type (Section IV), a detailed description of each process (Section V), special performance considerations/observations (Section VI), and summary comments (Section VII).

Early in the use of this system, the performance, ease of system management, and user friendliness were appreciated as major accomplishments and of significant value to the corporation. As time has progessed, another aspect has become apparent as perhaps even more valuable: the ability to use a single system design to control an application system regardless of the growth of the system. The design presented herein is effective whether used on a single application, a set of integrated applications, or all corporate applications. The initial investment in design has saved the corporation tens of thousands of dollars by preventing subsystem interfacing problems and hundreds of thousands of dollars because of performance/throughput characteristics.

The use of a single, common mechanism to interface the user with the application modules has expedited our ability to introduce new applications. At this point our users are familiar with the system since all applications use the same conventions and techniques. This in turn minimizes training costs and promotes greater accuracy in the information being input to the system.

So, if you are developing application systems and are interested in any of the above results, venture forth to Section II.

II. REQUIREMENTS

The decision to consider process handling as a technique to help us achieve our goal is more understandable when one considers the specific requirements and environment preceding our decision. Approximately five years ago, the decision was made to convert a large scale on-line manufacturing/accounting system to use HP3000 computers. The requirements are summarized as follows:

1. No user MPE knowledge required.
2. Data would be divisionalized on multi-CPU's.
3. Five second or less median response time.
4. Application scope, extensive and expandable.
5. On-line inquiry/update.
6. Audit/Security provisions.
7. Not menu driven.
8. Logical transaction recovery capability.
9. Data must use a DBMS (IMAGE/3000)
10. Terminals must be block mode.

Of the aforementioned ten requirements, only the first four were significant in our decision to use process handling. See Appendix 1 for additional information on the related requirements 5 through 10. The first four are described as follows:

Requirement 1 -- No User MPE Knowledge Required

One objective was that users would not be required to learn any aspect of MPE, e.g., how to sign on, build files, use sort, run programs, know JCL or any other fun trivia that can only diminish your profit sharing check. Users are trained on how to used the application commands (screens) associated with the functional requirements of their job.

With this as a requirement, one could either have one process handling all terminals (Would you bet your job that this would not produce a bottleneck?) or one process associated with each terminal. We chose the latter. Hereafter, the process associated with each terminal is called the "On-Line" process and the parent for all of these "On-Line" processes is hereafter called the "Controller" process. Thus, with this single requirement, process handling became a strong candidate for the fundamental processing design.

Requirement 2 -- Multi-Divisional/Multi-CPU Data

Shortly prior to conversion, the corporation had divisionalized into three manufacturing divisions with a central manufacturing support organization and corporate accounting/personnel functions. A corporate objective was to provide the capability for each division to independently control their data processing function while using a common set of software. The potential for each business unit (division) to do its processing on an independent CPU was considered desirable (mandatory by some).

A self-imposed requirement considered imperative by Data Processing was that data would not be added, changed or deleted by a process on a remote CPU. Using process handling, a mechanism was designed to transfer a user from one CPU to a second, if necessary, without the user having to understand network communications or any other aspect of inter-CPU communications. This process, hereafter referred to as the "Switching" process, requires that the user only needs to understand the application and determine which division's data is to be reviewed or changed. The multi-CPU environment becomes totally transparent to the user.

A second aspect of a multi-CPU environment was the desirability of being able to process on any CPU even if access to a remote CPU was unavailable. This leads one to consider what data, if any, should be redundant in a distributed or multi-CPU environment. Once it was decided to maintain copies of customer and vendor data on each CPU, the problem of selecting a mechanism to maintain concurrancy on the redundant copies had to be addressed. Our answer was to design a process, hereafter referred to as the "Pickup" process, which would pick up a copy of the original transaction from the originating remote CPU and pass it to a process, hereafter called the "Background" process, for processing. The "Background" process has the total capability of an "On-Line" process, except no terminal is associated with the process.

The parent process for "Switching", "Pickup" and "Background" is the "Controller".

Requirement 3 -- Five Second Response Time

Of all the requirements, the requirement to have a median response time of less than five seconds was the requirement of greatest concern. Response time was defined as the elapsed time between the user pressing the "ENTER" key and the next screen being displayed (returned) to the user.

Previous experience had shown that actions such as creating processes, opening/closing data bases and using mail for inter-process communication were expensive in resource utilization and were time consuming. Thus, our original design minimized these actions.

The next decision was to provide a mechanism by which control of the terminal could be returned to the user for entry of the next command while the previous command was being processed. To enable this approach to work, one must do all editing prior to passing the data to the "Background" process for updating. Background processing is also only utilized when the updating portion of the processing is significantly more extensive than the editing portion. Todate, only 38 of 900 commands pass data to be processed in the background mode; however, the CPU and wall time for these transactions account for 40% of the total CPU and wall time.

Requirement 4 -- Application Scope, Extensive & Expandable

As can be seen in Figure 1, which shows the current application scope, the major business and accounting applications associated with a manufacturing organization are included within the system. The data for these functions are included within a set of integrated data bases shown in Figure 2. No data is redundant among the application areas. From the list of applications, it is readily apparent that this is a mature system. However, it is still growing and the system design provides for the expansion of current application areas as well as integration with new applications!

Accounts Payable	Purchasing
Accounts Receivable	Provisioning
Air Traffic Association	Receiving
Bill-of-Material	Sales Analysis
Document Inventory	Sales Order Management
Engineering Change Orders	Security (DP)
Inventory Control	Shipping
Marketing	Shop Floor Dispatch
Material Requirements (MRP)	Spares Pricing
Master Scheduling	Standard Cost
Order Entry	User Documentation (DP)
Payroll	Work-in-Process
Personnel	Work Order Release
	Wirelists

Figure 1 -- Scope: Applications & Subsystems

Each Division:

DBAP	--	Accounts Payable
DBAR	--	Accounts Receivable
DBENG	--	Part Data, Part Lists, Routings, Notes, Documents
DBENGX	--	Engineering Change Orders, Equipment Data
DBHIST	--	Historical Order Data
DBMKT	--	Marketing Data
DBMRP	--	Material Requirements Planning
DBOR	--	Sales Orders, Forecast Orders, Manufacturing Orders
DBPO	--	Purchasing
DBRCP	--	DP Batch Processing Flags
DBWIP	--	Shop Orders, Work-in-Process, Work Centers, etc.

Each CPU:

DBCPU	--	DP Configuration Constants
DBCU	--	Customer Data
DBDOC	--	DP Documentation
DBEM	--	Payroll/Personnel
DBSEC	--	Security for DP System
DBTBL	--	Application Tables
DBVN	--	Vendor Data

Special Divisional Data:

DBATA	--	Air Traffic Association Data
DBPROV	--	Provisioning
DBWIRE	--	Wire List Data

Figure 2 -- Scope: Major Data Bases

III. ENVIRONMENT/CONFIGURATION

The environment of the current system is shown in Figure 3. The three CPU's (A, B and C) are Series 48's with 4 Mb of memory each with approximately 50 terminals each. A division resides on each of these. Sytem B also contains some corporate functions and central manufacturing services. System D is an HP3000 Series 42 with 3Mb of memory and approximately 20 terminals. It is allocated for development, testing and special user functions.

All computers are linked using DS3000 X.25 through a Memotec MPAC2500 switch. System C is connected to the switch using a 9600 baud telephone line with systems A, B and D being hardwired. Systems A and B are also hardwired using a 56Kb line. Twelve 7933 disc drives (404Mb each) are distributed among the CPU's. Each system uses a high speed tape drive for backup. A mixture of printers, micro's, plotters and other peripherals are attached to the various CPU's.

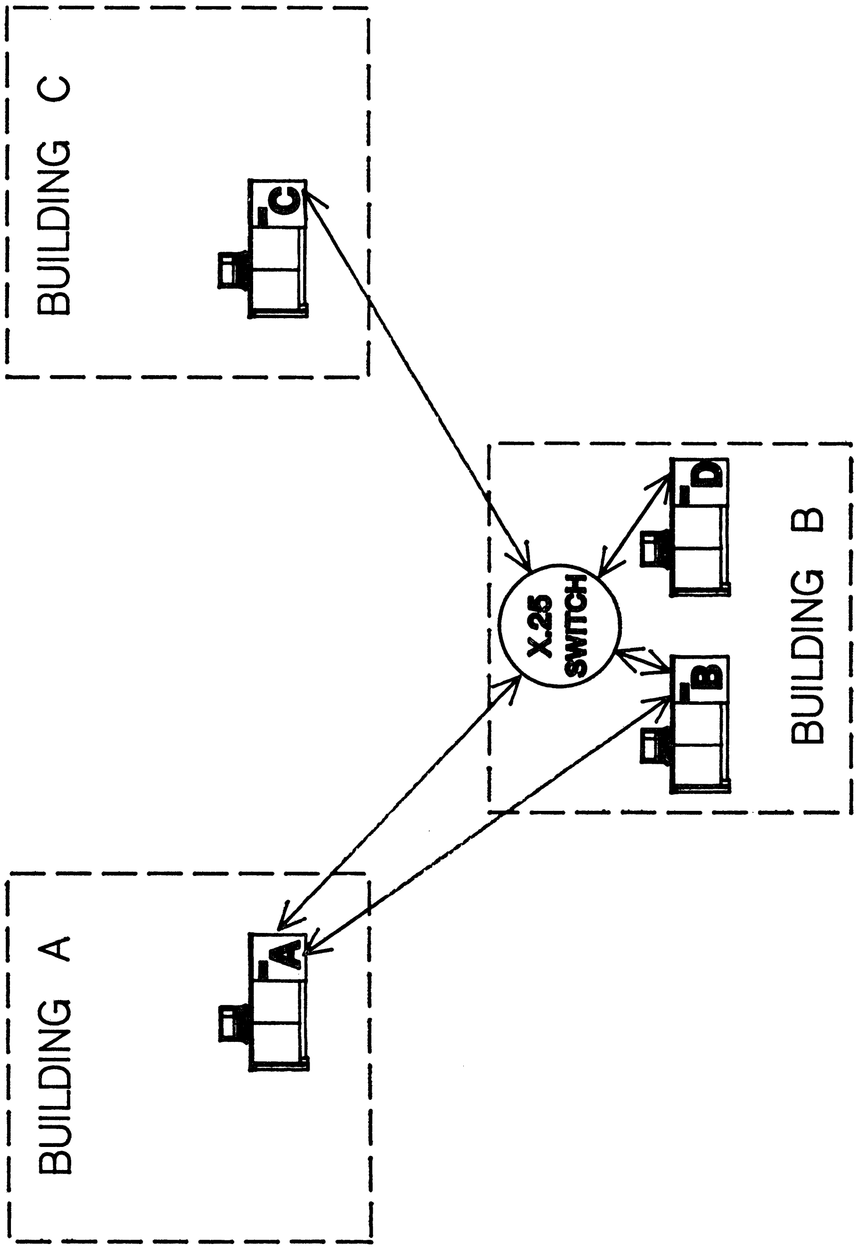


FIGURE 3 — ENVIRONMENT / CONFIGURATION

IV. THE PROCESS TREE (BAP)

From 7AM until 7PM, the process tree shown in Figure 4 is run on each of the primary CPU's (systems A,B and C). These business application processors, which are collectively referred to as "BAP", handle the on-line inquiry and updates for all the applications which are shown in Figure 1. A detailed description of each process is in Section IV of this paper. A brief summary of each process is as follows:

C-I Process: The BAP Command Interpreter accepts commands from a terminal called the BAP Console. Commands are typically to start, terminate or show the status of grandchildren processes. The process classes are: "On-Line", "Background", "Pickup" and "Switching". The "Controller" process is started automatically by the BAP "C-I". All other processes are created by the "Controller". The "C-I" communicates with the "Controller" via an extra data segment "COMSEG", which is described in Appendix 2.

Controller Process: The "Controller" is created by the BAP "C-I". It reads an extra data segment, "COMSEG", to determine what if any actions it is to perform. Typically, its functions are to create or terminate son processes. It may also send messages to the BAP "C-I".

On-Line Process: The "On-Line" process is created by the "Controller". It opens a terminal for use and posts a timed read against it. If data is read, it calls the appropriate application module after certain edits. When the transaction is completed, it logs the transaction, checks and updates "COMSEG" and posts another timed read. If no data is read, it checks "COMSEG" and then returns to a timed terminal read. If the user has not entered any data after a certain number of timeouts, the user is signed off. The process then initializes to a signon state.

Background Process: The "Background" process is created by the "Controller". It performs the same as the "On-Line" process except that it reads an IPC file for log record numbers to be processed.

Pickup Process: The "Pickup" process is created by the "Controller". It checks a remote file to see if data is present. If so, it reads the remote data, logs it to an input file for "Background" and posts the record number to the IPC file which the "Background" process reads.

Switching Process: The "Switching" process is created by the "Controller". It posts a timed read on an IPC file. If data is read, it modifies "COMSEG" and awakens the "Controller" which will create the process. It also checks "COMSEG" to see if any special action exists to be done such as terminating itself.

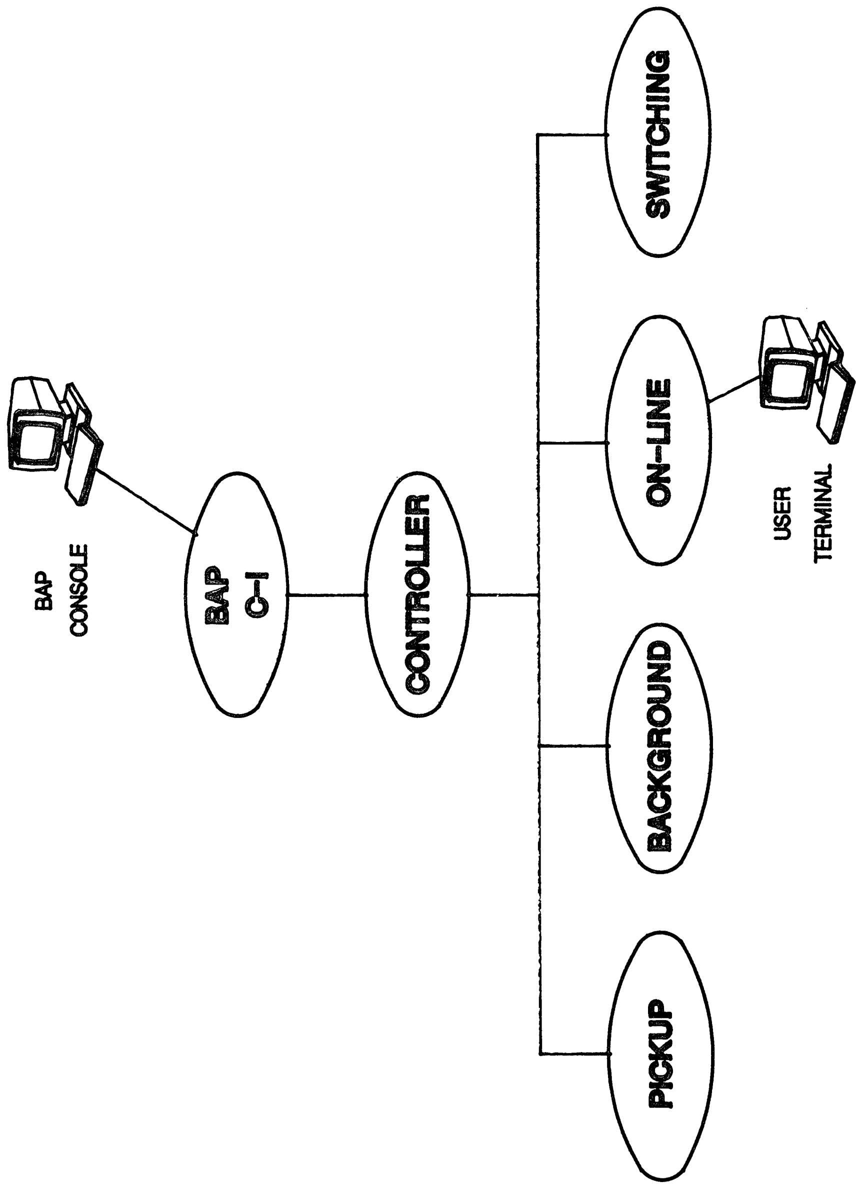


FIGURE 4 -- BAP PROCESS TREE

V. BAP PROCESSES

C-I Process

The function of the "C-I" is to accept commands from the BAP console. The BAP console is merely the terminal on which the "C-I" process is run. For each CPU, only one "C-I" process is run. It has the capability to control all activity in the BAP process tree. The principal functions of the "C-I" are to initialize files, startup processes, report status information and provide a clean shutdown. Auxiliary functions include sending messages to "On-Line" users and establish a new user signon welcome message.

The "C-I" process, shown graphically in Figure 5, performs the following specific functions:

1. Initializes logfiles.
2. Opens DS lines to remote CPU's
3. Creates extra data segment "COMSEG"
4. Opens background IPC file.
5. Creates "Controller" process.
6. Initializes welcome message to be used on "On-Line" signon.
7. Cycles on read terminal until exit. The following actions are fully described in Appendix 3: create, terminate, tell, welcome, status, kill, exit, pause.

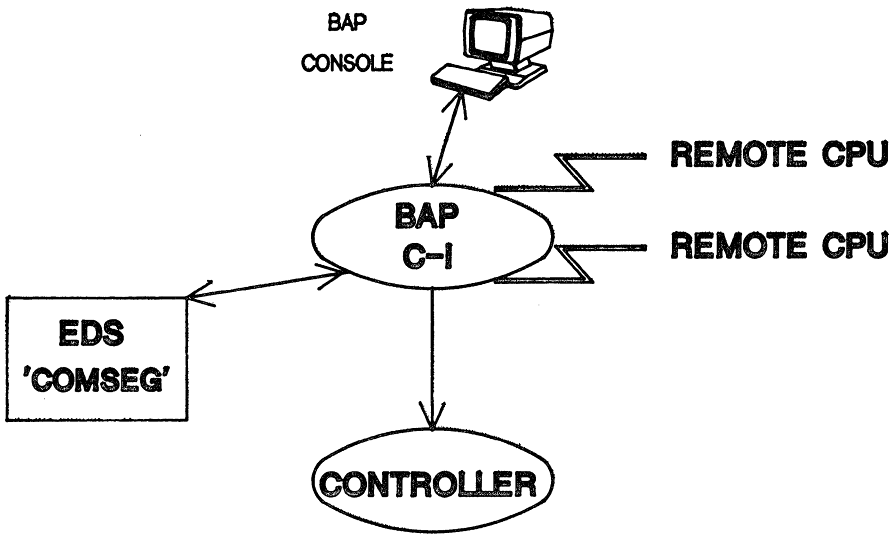


FIGURE 5 -- BAP C-I

Controller Process

When activated by either its parent, the "C-I", or by one of its children, the "Controller", shown in Figure 6, reads the extra data segment "COMSEG". See Appendix 2 for a description of "COMSEG". Once awakened, the "Controller" will perform all required actions indicated in "COMSEG" and then suspend itself. Actions include the following: create a process, kill a process, send a message to the BAP console, and update "COMSEG" appropriately.

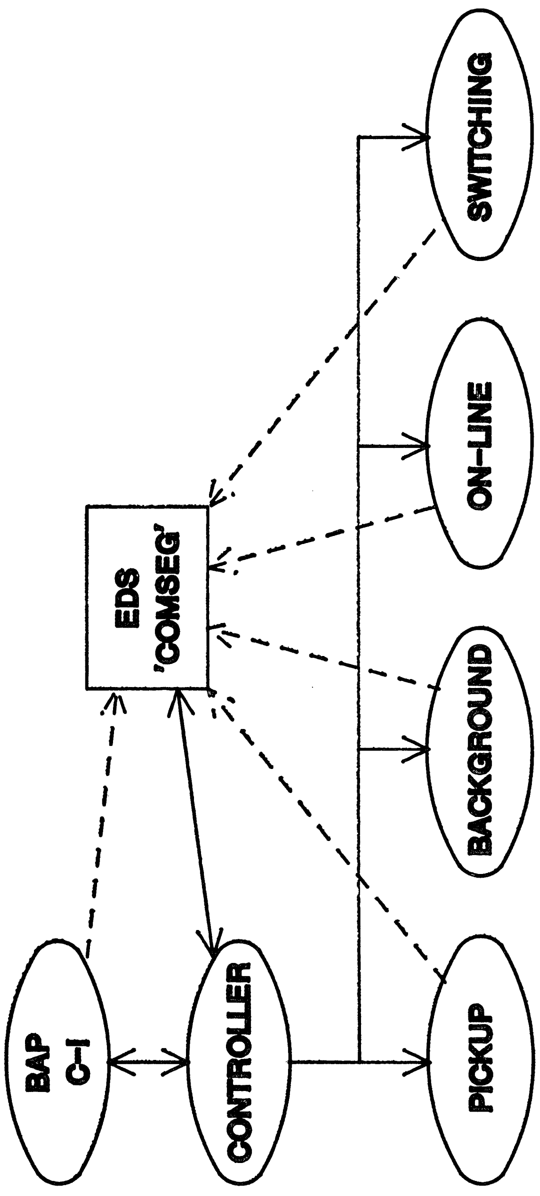


FIGURE 6 — CONTROLLER PROCESS

On-Line Process

The "On-Line" process, as shown in Figure 7, is created by the "Controller". It then performs as follows:

1. Initializes global data and data to be shared with called application modules.
2. Initializes data base file count.
3. Initializes log record.
4. Opens diagnostic file for error messages, fatal error dump information.
5. Opens message catalog.
6. Fetches global constants.
7. Sets read timeout.
8. Opens security database.
9. Opens user terminal in block mode.
10. Fetches formfile names.
11. Opens logfiles (See Appendix 4 for log record layout)
12. Opens background IPC file.
13. Opens background IPC record available file.
14. Cycles on terminal read until "EXIT".
    a. If first pass, displays signon screen.
    b. Checks "COMSEG" and performs appropriately.
    c. If "BYE", signs off user and displays signon screen.
    d. If "EXIT", closes terminal and terminates process.
    e. Validates command-id.
    f. Checks user security.
    g. If security fails, falls through to step 14.o
    h. Logs transaction starting.
    i. Calls application module (PCAL)
    j. Logs transaction completing.
    k. If partial update, terminates this process.
    l. If major error, flags comseg to terminate all processes.
    n. If multi-CPU, posts to all CPU's.
    o. Posts timed read on user terminal.
15. If no data was read but timeout occurred, checks timeout provisions. If necessary, signs off user and returns to 14-a.
16. If user signed on to remote CPU, updates "COMSEG", activates the controller and terminates this process.

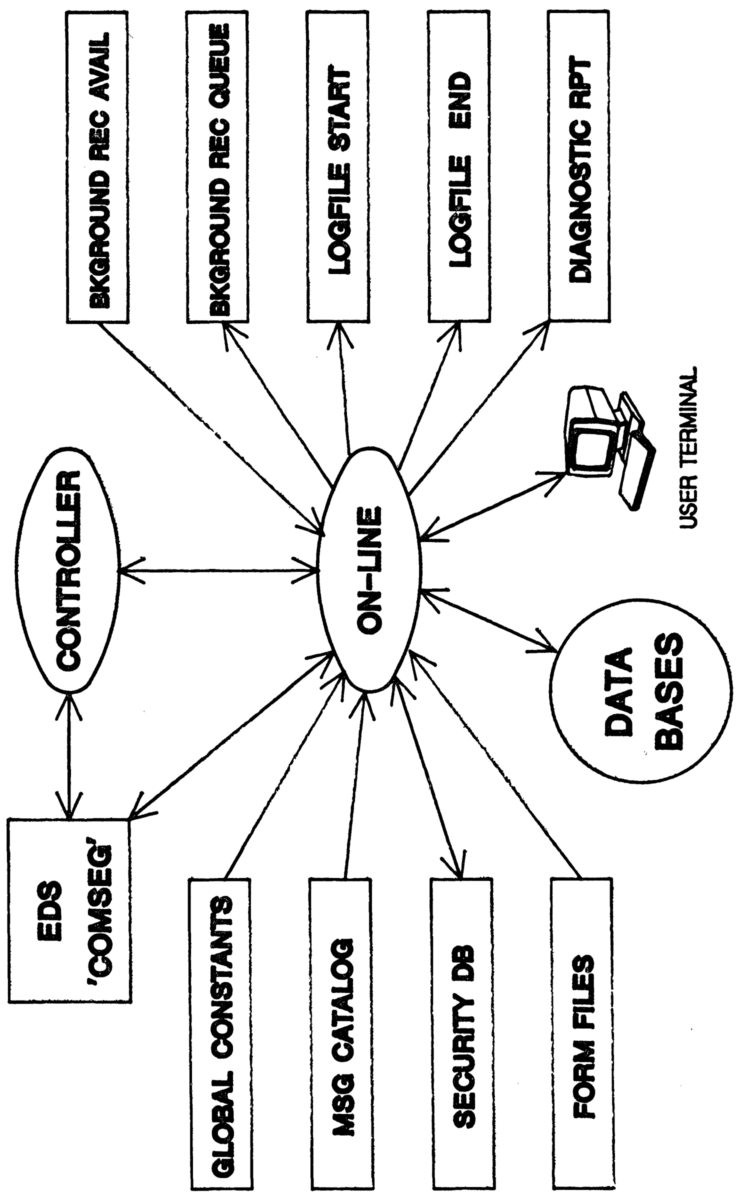


FIGURE 7 — ON-LINE PROCESS

Background Process

The "Background" process, as shown in Figure 8, is created by the "Controller". It then performs as follows:

1. Initializes global data and data to be shared with called application modules.
2. Initializes data base file count.
3. Opens diagnostic file for error messages, fatal error dump information.
4. Opens message catalog.
5. Fetches global constants.
6. Opens security database.
7. Opens logfiles (start & end).
8. Opens IPC file for log record number to be processed.
9. Opens IPC file for reusable log record numbers.
10. Cycles on timed read of IPC file of log records to be processed.
    a. Checks "COMSEG" and performs appropriately.
    b. Reads record to be processed from LOGFILE1.
    c. Validates command-id.
    d. Logs transaction starting.
    e. Calls application module.
    f. Logs transaction completing.
    g. If partial update, terminates this process.
    h. If major error, flags comseg to terminate all processes.
    i. If multi-CPU, posts to all CPU's.
    j. Updates IPC file for reusable log record numbers.
    k. Posts timed read on IPC file of log records to be processed.
11. When terminate flag is set in "COMSEG", process terminates itself.

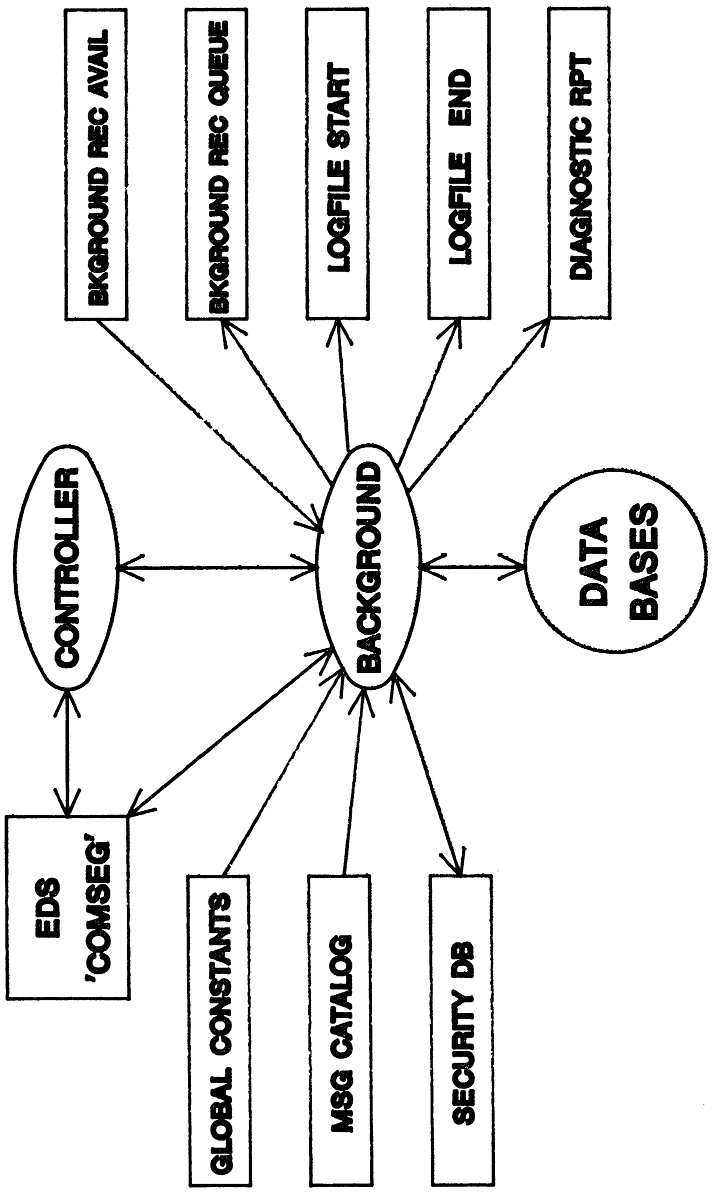


FIGURE 8 — BACKGROUND PROCESS

Pickup Process

The function of the "Pickup" process is to read records posted to files on remote systems and to pass the transactions to "Background" to be processed. A "Pickup" process is created by the "Controller" for each remote CPU. Another approach would be to have a single process service all remote pickup files. The "Pickup" process, shown in Figure 9, performs as follows:

1. Initializes global data.
2. Opens diagnostic file for error messages, fatal error dump information.
3. Opens message catalog.
4. Fetches global constants.
5. Opens security data base.
6. Opens logfiles.
7. Opens background IPC file.
8. Opens background IPC record available file.
9. Issues file equations for remote pickup files.
10. Opens remote pickup files.
11. Cycles on checking "COMSEG".
    a. Determines if any records exist to be picked up.
    b. If records exist, does the following:
        i. Reads record.
        ii. Passes record to "Background".
        iii. Goes to 11-a.
    c. Pauses global pause time.
    d. Checks "COMSEG".
12. When terminate status in "COMSEG" is set, process terminates itself.

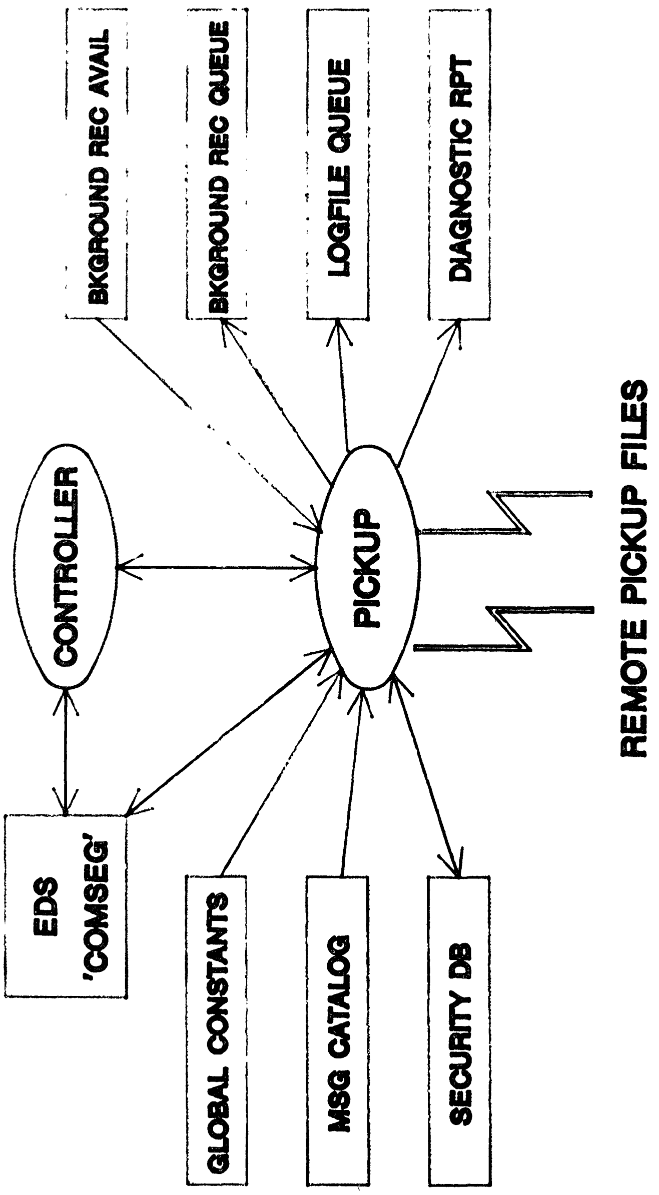


FIGURE 9 -- PICKUP PROCESS

Switching Process

The "Switching" process is created by the "Controller". The function of switching is to terminate a user's process on the current CPU and initiate an "On-Line" process on a different CPU. This is required whenever a user is signing on to a division which is not on the current CPU. The transfer routing is controlled by a directory so that the user has no requirements other than specifying the destination division. The "Switching" process, shown graphically in Figure 10, performs the following specific functions:

1. Initializes global data.
2. Opens diagnostic file for error messages, fatal error data.
3. Opens message catalog.
4. Fetches global constants
5. Opens transfer request IPC file.
6. Cycles on timed read of transfer request file.
   a. Checks "COMSEG" and perform appropriately.
   b. Builds a "COMSEG" device entry record.
   c. Pauses to let process which issued request terminate itself.
   d. Puts device entry in "COMSEG"
   e. Displays message to "C-I".
   f. Activates "Controller" who will create the process.
   g. Issues timed read on transfer request file.
7. When terminate flag is set in "COMSEG", process terminates itself.

To better understand the "Switching" process, Figure 11 contains a step by step diagram which is described as follows:

Step 1 -- User signs on to division which is not on current CPU. The "On-Line" process determines which CPU is the correct one and posts a record in the switch data file.

Step 2 -- "On-Line" driver sets termination flag in "COMSEG".

Step 3 -- "On-Line" driver activates the "Controller".

Step 4 -- "Controller" terminates "On-Line" process.

Step 5 --"Switching" process on remote CPU reads switch data file.

Step 6 --"Switching" process posts data to "COMSEG".

Step 7 --"Switching" process activates remote "Controller".

Step 8 -- Remote "Controller" creates issues a file equation for the user terminal and creates an "On-Line" process.

Step 9 -- The remote "On-Line" process displays the next screen.

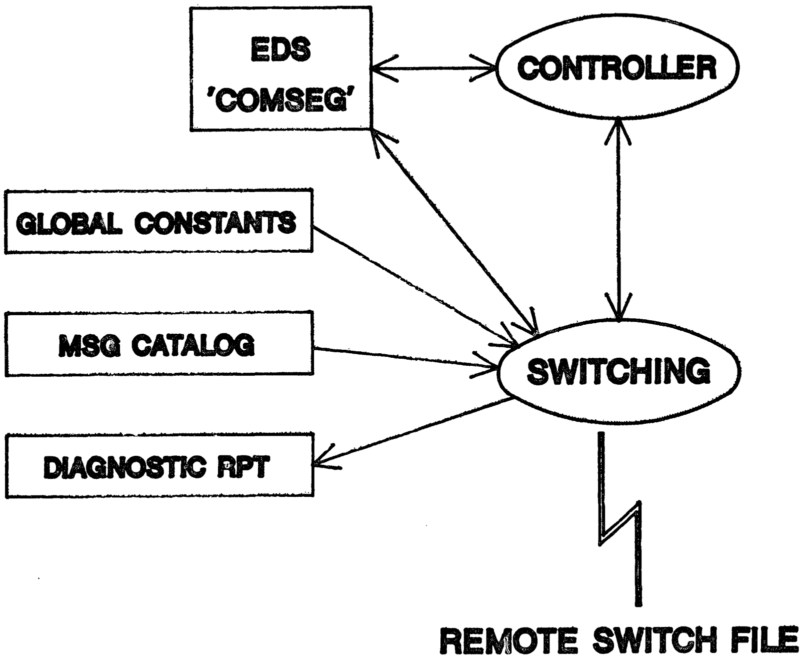


FIGURE 10 –– SWITCHING PROCESS

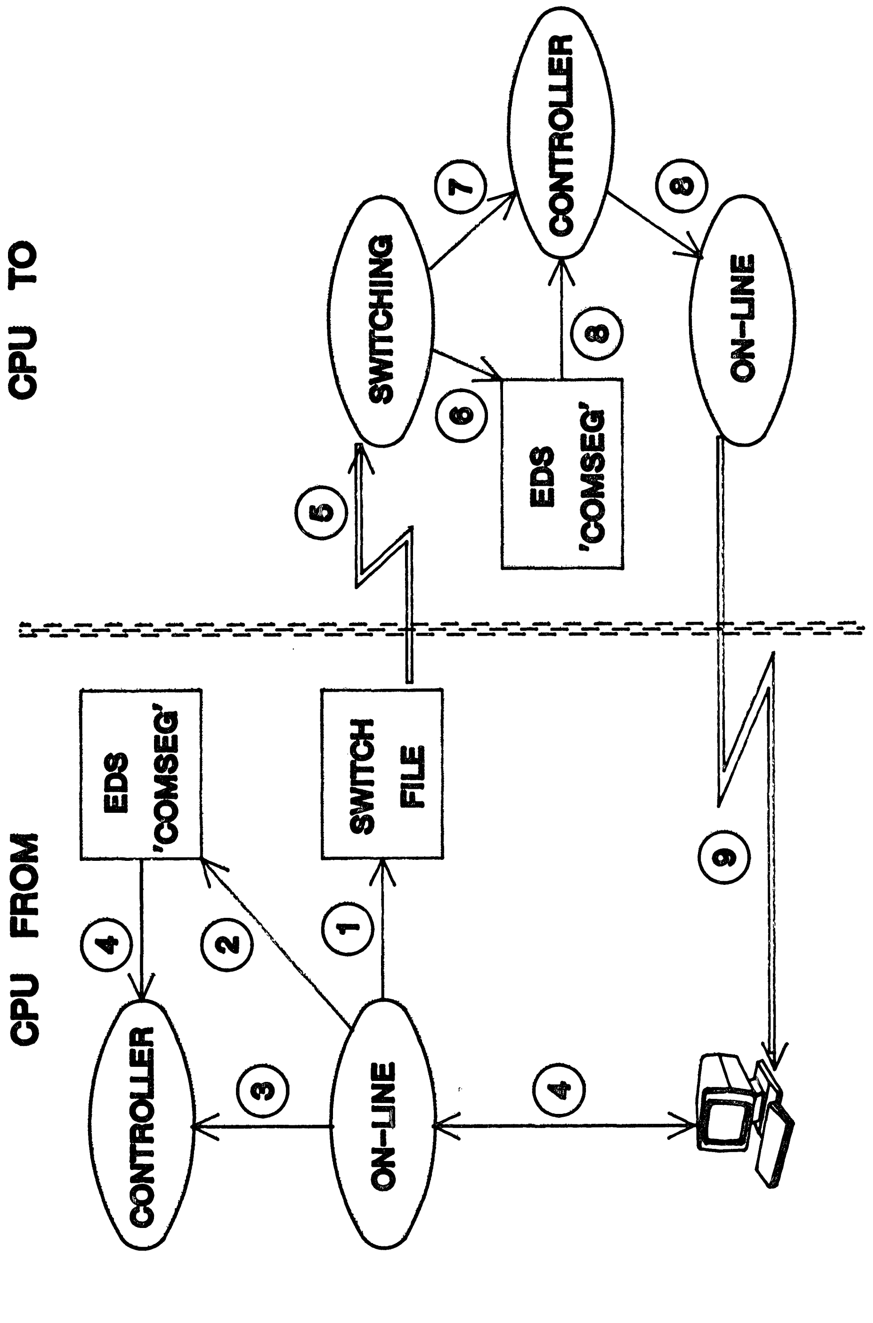


FIGURE 11 — PROCESS SWITCHING (STEPS)

V. PERFORMANCE CONSIDERATIONS

The first principle used in our design was to minimize the number of entities the system (MPE) has to keep track of. For example, the use of a process tree enables one session to handle many terminals. In our environment, it is common for thirty-five "On-Line" processes to be in the process tree at any one time. Thus, MPE has to only keep track of one session, not 35.

Another entity is the use of DS lines for communication with the remote CPU's. By having a process tree, these lines are opened only once and only one remote session is established per CPU.

The other side of this coin is that a bottleneck can be established. The reply to this is that process handling provides for any number of lines to be opened. In fact, it is possible to design load leveling algorithms to select the currently unused entity. The point is that the decision of how many copies of an entity are active is left to the descretion of the system designer/manager.

The second priciple used was to minimize actions, e.g., opening files, opening data bases, fetching certain data, etc. This provides a significant improvement in system response.

The third principle was to provide a large library of callable routines in segmented libraries (SL's) so that the system would have less code to manage and would use less memory for the application code. This is not directly related to process handling, but if you're still reading at this point, you probably need all the help you can get.

The fourth priciple was to return control of the terminal to the user as soon as the transaction was assured of being able to complete. To use this concept, all edits must be done prior to any update. From this priciple, the concept of a background process was developed. To be effective, the amount of work to be done after edits has to be significantly large. For example, a transaction which only updates a single record would not be considered for background processing, whereas a transaction which causes fifty records to be updated would be. In other words, the reduction in user wait time should be significant since additional overhead is being incurred by submitting a transaction for background processing.

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

VI. SUMMARY

The use of process handling to drive our application code has fulfilled all of our requirements and surpassed our expectations in being adaptable to growth. By possessing such a commonality in approach, implementation of new applications has been expedited and training minimized. The consistancy in approach has also improved the accuracy of information entering the system.

The particular design presented in this paper is just one way of using process handling to optimize system management and system throughput based on a particular set of specific requirements. This design could easily be modified to accomodate special situations or unique load requirements. Alternative designs could just as easily be developed.

The use of process handling allows the system designer to provide for unique situations, which frequently only become apparent after significant growth, without having to modify each program in the system. Thus, for any installation which is developing a significant amount of code, or is anticipating development over an extended time period, I would recommend that careful consideration be given to the use of process handling as a design approach.

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 1

REQUIREMENTS ASSOCIATED WITH THE SYSTEM

Requirement 5 -- On-Line Inquiry/Update

The system prior to conversion processed data inquiries, updates, additions and deletions on-line. Although the system also processed a significant nightly batch which did both reporting and updating, the long range objective was to eliminate batch and do all updating on-line. Thus the anticipated load was expected to increase significantly during the day, even without increased business activity or expanded applications.

Requirement 6 -- Audit/Security Provisions

Security was a requirement at the employee, division, and command level. For example, an employee might be authorized to execute an update command in his own divsion, review similiar data in a second division and do neither in a third division. The security is controlled by division/corporate controllers.

The audit requirements were that each logical transaction was to be tagged with the user-id , time of transaction and terminal number. Each record in the data base is timed stamped such that the last user to change the data can be identified. Each logical transaction is logged to a permanent file from which performance and usage statistics are generated.

Requirement 7 -- Not Menu Driven

Although on-line help and documentation were desired, it was considered a critical requirement that a user could execute any authorized command in any sequence without having to meander through a series of menus or specific sequence of screens just to get to the screen the user desired to use. Considering that close to 900 commands exist today (backed by 1.2 million lines of source code and over 2Mb of on-line executable code), this requirement played a major role in the design of our process handling system. Note that a large amount of application code had to be accessible from any "On-Line" process with no observable access time difference between commands. Access also had to be quick enough to also meet the minimum transaction time requirement.

APPENDIX 1 -- Continued

Requirement 8 -- Logical transaction recovery capability
==

In the previous system, each screen transmitted from the user's terminal contained a command-id and associated data. Each of these transactions contained all the data necessary for processing; i.e., each transaction stands alone. In the event of the necessity of having to run recovery (Have you ever had a head crash?), the transactions which completed successfully were reprocessed. Obviously, much more could be said about backup, recovery approaches, logging and related subjects, but this paper is about process handling.

Requirement 9 -- Data Must Use a DBMS (IMAGE/3000)
==

The previous system was based upon an integrated set of linked files. To facilitate system change, growth and user inquiry using fourth generation tools, all data is contained within IMAGE/3000 data bases.

Requirement 10 -- Terminals Must Be Block Mode
==

The previous system was based on a block mode terminal. V/3000 was selected for all screen handling but no editing. Why not?

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 2

COMSEG

COMSEG, an extra data segment, is a means by which the BAP processes on one CPU can share data. It consists of a header, a device table, a message table, and the WELCOME message.

```
header
device entry #0      }
device entry #1      }
      .              }
      .              } device table
      .              }  (one entry for each
                     }   UT020P son process)
device entry #n      }

message entry #0     }
      .              }
      .              } message table
      .              }
                     }
message entry #9     }
WELCOME message
```

Header Format

Word #	Data Type	Contents
0 - 1	double	log record number
2 - 3	double	log file size
4		
5	integer	address of device table
6	integer	device entry size (words)
7	integer	device table size (entries)
8	integer	address of message table
9	integer	message entry size (words)
10	integer	message table size (entries) maximum = 10
11	integer	address of WELCOME messages

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 2 -- Continued

Device Entry Format

Word #	Data Type	Contents
0	integer	logical device number of forms terminal (0 if this entry is empty)
1	logical	true if DEBUG to be used, otherwise false
2 - 4	ASCII	current user ID (blank if no one signed on)
5 - 6	ASCII	current ORG-ID
7 - 9	ASCII	current transaction
10(1:15)	integer	current user process status (UPSTATUS) 0 - create pending interpreter 1 - created, active pending 2 - active 3 - terminate pending 4 - terminated 5 - disable pending 6 - disabled 7 - enable pending 8 - enabled 9 - abort pending 10 - aborted 11 - transferred (to another CPU) 13 - kill pending
10.(0:1)		on if message not yet sent to operator indicating change of status
11	integer	PIN for this process
12 - 21	integer	message indices (-1 for none)
22	logical	origin (0 - command interpreter >0 - another process structure)
23 - 26	ASCII	terminal's home CPU-ID
27 - 30	ASCII	name of CPU this terminal is transferring to
31 - 36	ASCII	user ID and password (transfer signon info)
37 - 38	ASCII	ORG-ID (transfer signon info)

Message Entry Format

Word #	Data Type	Contents
0	integer	usage (# users yet to receive this message)
1 - 39	ASCII	message

APPENDIX 2 -- Continued

Welcome Message Format

Word #	Data Type	Contents
0 - 38	ASCII	line 0 of welcome message
.	.	.
.	.	.
.	.	.
.	.	line 19 of welcome message

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 3

C-I COMMANDS

```
CREATE list [;DEBUG]
====================
```

Puts a device entry into "COMSEG" and activates the "Controller". The "Debug" option causes the process to be created with Debug.

```
TERMINATE {list} [;CPU'ID]
          {ALL }
==========================
```

Sets the status flag in each device entry in the list and activates the "Controller". The "ALL" option sets the flag in all device entries. The "CPU'ID" option sets the flag in the device on the specified CPU. Note that in multiple CPU environments, more than one device entry could have the same logical device number; thus, for remote devices, the CPU'ID is also given.

```
KILL {list} [;CPU'ID]
     {ALL }
=====================
```

Sets the status flag in specified device(s) to kill pending and activiates the "Controller".

```
EXIT
====
```

Terminates the "C-I" which causes all descendent process to be terminated.

```
STATUS {list} [;CPU'ID]
       {ALL }
=======================
```

Read each device entry in "COMSEG" and prints a line of information on the BAP console. Also shows the remaining number of background transactions.

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 3 -- Continued

```
PAUSE {number'of'minutes}
=========================
```

Calls the PAUSE intrinsic for the specified number of minutes. Used in conjunction of running a process in DEBUG.

```
TELL {list} [;CPU'ID]
     {ALL }
=====================
```

Send a message to the device(s) in the list. Requests up to ten lines of messages from the BAP console user. Puts this message in the message table of "COMSEG". Each "On-Line" process checks after each transaction or time-out to see it messages exist. If so, it displays the message to the "On-Line" user terminal.

```
WELCOME
=======
```

Updates the welcome message in "COMSEG" from a standard file. This message is displayed on the "SIGNON" screen.

USING PROCESS HANDLING TO OPTIMIZE SYSTEM PERFORMANCE

APPENDIX 4

LOGFILE FORMAT

Character	Data Type	Description
0-1	ASCII	Year
2-3	ASCII	Month
4-5	ASCII	Day
6-7	ASCII	Hour
8-9	ASCII	Minute
10-11	ASCII	Second
12	ASCII	Forms terminal CPU }
13-15	ASCII	Forms terminal ldev# }orgntr
16-20	ASCII	User-id }
21-25	ASCII	CPU time for this transaction (tenths of a second)
26-31	ASCII	Wall time for this transaction (tenths of a second)
32-35	Double	Lock stamp (milliseconds)
36-37	ASCII	Recovery code "Q","S","P","N","C","I","L"
38-41	ASCII	ORG-ID
42-43	ASCII	Error code (UXECODE value or 4 if edit error detected)
44-45	Integer	Transaction length (bytes)
46-51	ASCII	Current form name (prior to calling application)
52	-	This CPU-ID
53	ASCII	"B"=executed in background, space = normal
54-57	ASCII	Originating org (indicates redundant request if not blank)
58-59	-	Unused
60-65	ASCII	Transaction command-id (1st 6 char of transaction)
66-1987	ASCII	Transaction data (Variable length)
1988-2047	--	Unused

3055. Everything You Wanted to Know about Interfacing to the HP-3000 The Latest Inside Story

Ross Scroggs
Telamon, Inc.
1615 Broadway, 11th Floor
Oakland, CA 94612
(415) 835-5603

INTRODUCTION

The subject of terminal interfacing to the HP-3000 contains no facts. None. Everything I say and you observe is an illusion supported by a lack of information and the general perversity of the universe. Maybe terminal interfacing is the fourth dimension, moving through it is certainly stranger than anything you have ever experienced before. But the universe is becoming stable, and the strangeness is beginning to abate, or I've finally become immune to change.

I have included a list of references at the end of this paper from which I have obtained some of the information included here. If you desire to make all of your terminal attachments successful, obtain all of the references and read them. The most important piece of information I can give you is to start planning early when attaching terminals to the HP-3000 and don't believe anything you read, including this paper. If you haven't seen it work yourself, plan on having to solve a few problems. This paper, derived from eleven years of HP-3000 experience, is a guide to solving those problems, but it won't solve them for you.

Asynchronous terminals are attached to the HP-3000 Series I, II, and III through the Asynchronous Terminal Controller (ATC); to the Series 30/33/39 and 40/42/44/48 through the Asynchronous Data Communications Controller (ADCC); and to the Series 42/44/48 (optional), Series 37 and 64/68 through the Advanced Terminal Processor (ATP). This paper addresses issues involved in making a successful connection to one of these three devices.

The experiments in this paper were conducted on a Series III with ATCs, a Series 42 with ADCCs running T Delta 1, and a Series 37 with ATPs running T Delta 1. You should expect that your results will differ when using different machines and operating system releases.

RS-232 and RS-422 are standards which describe an interface specification. They describe the electrical characteristics and control signalling conventions used by devices conforming to the standard. They do not guarantee that two RS-232 devices can communicate with each other. It is the user's responsibility to ensure compatibility of devices at the data level. The principal focus of this paper is the description on the factors that the user must control.

Terminals attached to the HP-3000 are accessed in two ways: as a session device or as a programmatically controlled device. A session device is one on which a user logs on with the HELLO or () commands and accesses the HP-3000 through MPE commands. A programmatic device is one which is controlled by an application program that is run independently from the device. These two access methods are not mutually exclusive; a session device can be accessed programmatically and many MPE commands can be executed on behalf of a user who is accessing the system programmatically.

SESSION DEVICES

Attaching a terminal as a session device is typically the easier of the two methods. You must set the terminal speed, parity, subtype, and termtype correctly and provide the proper cable to complete the hookup.

Terminal Speed

The speeds supported by the ATC are: 110, 150, 300, 600, 1200, and 2400 baud. The speeds supported by the ADCC are those of the ATC plus 4800, 7200, and 9600 baud. The ATP additionally supports 19200 baud, but deletes 150 baud. Ports are either speed sensing or speed specified. Speed sensing ports automatically adjust the baud rate based upon the initial carriage return received. Speed specified ports require that the initial carriage return be received at the specified speed. Speed specified is in a state of limbo on ADCCs. HP is trying to eliminate it, as it is no longer required, but bugs in the speed sensing algorithm have kept it around. The ATP does not support speed specified ports; but this presents no problem as speed sensing works at all speeds.

Terminal Parity

The format of characters processed by the HP-3000 is: a single start bit, seven data bits, a parity bit, and one stop bit (two at 110 baud). The parity bit is: always zero (called space parity), always one (called mark parity), computed for odd parity, or computed for even parity. A character with eight data bits must have no parity bit to be compatible with the HP-3000. In this case, the eighth data bit must be set to a zero, as the HP-3000 will try to interpret it as parity even though the terminal considers it data.

Choosing the proper parity setting has been complicated by differences between the ATC and ADCC/ATP. The ATC inspects the parity bit of the initial carriage return received from the terminal and sets parity based on that bit. If the bit is a zero, the ATC generates odd parity on output. If it is a one, the ATC generates even parity on output. In either case the parity of incoming data is ignored and the parity bit is always set to zero before the data is passed to the data buffer. The ADCC and ATP also set parity based on the parity bit of the initial carriage return, but they do so with a slight, nasty twist. If the bit is a zero, the ADCC/ATP pass-through the parity bit supplied by the application program on output. If the initial parity bit is a one, the ADCC/ATP generate even parity on output. If pass-through parity was selected, the parity bit of the incoming data is passed through to the data buffer. If even parity was selected, the input data is checked for proper even parity and the parity bit is set to zero before the data is passed to the buffer. Thus, you can not use odd or mark parity on the ADCC/ATP. The odd parity will be interpreted as pass-through and the parity bits will wind up in your data buffer, wreaking havoc. Mark parity will be interpreted as even and all input will cause parity errors.

Subtype

Subtype specifies the type of connection between the terminal and the HP-3000. The principal choices are: direct connect, full duplex modem connect, and half duplex modem connect. The subtype also specifies if a terminal is to be speed sensed, or speed specified. The ATC supports subtypes 0 through 7, the ADCC supports subtypes 0 through 5; the ATP supports subtypes 0 and 1. Subtype 0 is used for directly connected terminals, no modem is used. Note that terminals that are attached to multiplexors can fit in this category, the modem involved is managed by the multiplexor, not the HP-3000. Subtype 1 is used for terminals connected through full duplex modems such as Bell 103, 212 and Vadic 34xx. Subtype 2 and 3 are used for terminals connected through half duplex modems such as the Bell 202S. Subtypes 0 through 3 speed sense on the initial carriage return. Subtypes 4 through 7 correspond to 0 through 3 with the difference that terminals using these subtypes will not be speed sensed; they will run at a specified speed that is set at configuration time. This subtype is often used to prevent the HP-3000 from trying to speed sense garbage which sometimes occurs when using short-haul modems (line-drivers) that do not have a terminal attached to the other end. The ATC can lock out ports when this problem occurs.

Termtype

Termtype specifies the characteristics of the terminal to be attached to the HP-3000. Most termtypes were derived from specific models of terminals that were attached to the HP in the old days, early 1970's. HP is changing this term to port protocol type. The ATC supports termtypes: 0-6, 9-13, 15, 16, 18, 19, and 31. The ADCC supports termtypes: 4, 6, 9, 10, 12, 13, 15, 16, 18 and 19/20/21/22. The ATP supports termtypes: 6, 9, 10, 12, 13, 15, 16, 18, and 19/20/21/22.

Termtype 4 is for Datapoint 3300 terminals, it outputs a DC3 at the end of each output line and responds to backspace with a control-y. Termtype 4 on the ADCC does not output DC3s at the end of each line.

Termtype 6 is the general non-HP hardcopy terminal type. It outputs a DC3 at the end of each line but responds to a backspace with a linefeed. The linefeed is on the first backspace of a series, this allows you to type corrections under the incorrect characters.

Termtype 9 is the general non-HP CRT terminal type. No DC3s are output at the end of the line and nothing strange happens on backspace, the cursor backs up just as you would expect. (The ATC strips out some escape sequences from the input stream that were generated by the CRT on which termtype 9 was patterned.)

Termtype 10 is the general HP CRT terminal type. It is characterized by the ENQ/ACK flow control protocol.

Termtype 13 is typically for those terminals at a great distance from the HP-3000 for which some local intelligence echos characters and the 3000 should not. (Telenet and Tymnet charge you for those echoed characters, that's reason enough not to have the HP-3000 echo them.)

Termtypes 15 and 16 are for HP-263x printers. Like termtype 10, the ENQ/ACK flow control protocol is used. When the HP-3000 sends the ENQ character and no ACK is received within a few seconds, another ENQ is sent. This repeats until an ACK is received. This mechanism ensures that no data is sent to a non-responding printer. Non-HP printers that support an answerback capability can make use of this feature. Configure them to return an ACK when an ENQ is received and the problem of output being sent to powered-off printers is eliminated.

Termtype 18 is just like termtype 13 except that no DC1 is issued on a terminal read.

Termtypes 19/20/21/22 are for spooled 2361B printers.

Certain termtypes less than 10 specify a delay after carriage control characters are output to the terminal. The ATC and ATP handle this by delaying for a certain of character times but do not output any characters. The ADCC actually outputs null characters. The most extreme case is termtype 6 which causes 45 nulls to be output after a CR/LF at 240 cps.

There is a new product from HP (and a contributed program) that allows ADCC/ATP users to construct custom terminal types. The Workstation Configurator allows a terminal type to be constructed that should match almost any specific terminal. The configuration is stored in a "termtype" file and HP supplies the default files. (Listf TT@.PUB.SYS to list them.) There are additional capabilities provided that have not been supported previously. For example, multiple line terminators can be defined to be equivalent to carriage return. A termtype could be set up to specify tab as a line terminator which would enhance the use of the ten-key pad on HP terminals for data entry operators.

Cable

Directly connected terminals (subtypes 0 and 4) use only three signals in the cable: pin 2 (Transmit Data), pin 3 (Receive Data), and pin 7 (Signal Ground). Note that all signal names are given from the point of view of the terminal, not the modem or the HP-3000, which acts like a modem. Typically the cable will connect: pin 2 at the terminal to pin 2 at the HP-3000, pin 3 at the terminal to pin 3 at the HP-3000, and pin 7 at the terminal to pin 7 at the HP-3000. This is not to say that your terminal does not require other signals, it just says that the HP-3000 is not going to provide them for you. If your terminal requires signals like Data Set Ready, Data Carrier Detect, or Clear To Send, you can usually supply these signals to the terminal with a simple cable patch. Jumper pin 4 (Request To Send) to pin 5 (Clear To Send). Jumper pin 20 (Data Terminal Ready) to pin 6 (Data Set Ready) and pin 8 (Data Carrier Detect). These two jumpers cause the terminal to supply its required signals to itself.

Modem connected terminals (subtypes 1 and 5) use seven signals in the cable: pin 2 (Transmit Data), pin 3 (Receive Data), pin 4 (Request To Send), pin 6 (Data Set Ready), pin 7 (Signal Ground), pin 8 (Data Carrier Detect), and pin 20 (Data Terminal Ready). Naming the signals gets complicated since the HP-3000 is acting like a modem and it is being attached to a modem. Typically, the cable that connects the HP-3000 to the modem will connect: pin 2 at the modem to pin 3 at the HP-3000, pin 3 at the modem to pin 2 at the HP-3000, pin 4 at the modem to pin 8 at the HP-3000, pin 6 at the modem to pin 20 at the HP-3000, pin 7 at the modem to pin 7 at the HP-3000, pin 8 at at the modem to pin 4 at the HP-3000, and pin 20 at the modem to pin 6 at the HP-3000.

You should note an important characteristic of the cable descriptions given above. The terminal to HP cable is "straight-through," with like-numbered pins connected together. The modem to HP cable is a "cross-over," with pairs of pins cross-connected. Why the difference? The explanation is that the world is divided into Data Terminal Equipment (DTE) and Data Communication Equipment (DCE). A DTE is a terminal or something like it which sits at the end of a data communication line. A DCE is a modem or something like it which is part of the data communication line. The distinction is not rigid as the HP-3000 acts like a DCE. A multiplexor may look like a DCE to the terminals attached to it, and like a DTE to the modem to which it is attached. The cabling principle is that a DTE to DCE connection uses a straight through cable and a DTE to DTE or DCE to DCE connection uses a cross over cable.

The cable that attaches your terminal to a modem should be specified in your terminal owners manual, consult it for proper connections.

Flow Control

Flow control is the mechanism by which the rate of data flow between the HP-3000 and the terminal is controlled. The HP-3000 supports two output flow control methods, ENQ/ACK and XON/XOFF. The HP supports one input flow control protocol, DC1/DC2/DC1, commonly referred to as the "block mode" protocol.

The ENQ/ACK protocol is controlled by the HP-3000. After every 80 characters output the system sends an ENQ to the terminal and suspends further output until and ACK is received back from the terminal. The suspension is of limited duration for termtypes 10 to 12, output resumes if no ACK is received in a short amount of time. The suspension is indefinite for termtypes 15 and 16, the ENQ is repeated every few seconds until an ACK is received.

It is the ENQ/ACK protocol that fouls up non-HP terminals attempting to access the HP-3000 through an ATC port configured for an HP terminal. Most terminals do not respond to an ENQ with an ACK, you must do it manually; type control-f which is the ACK. An ENQ is output by the ATC upon receipt of the initial carriage return from the terminal. You get hung immediately, unless you type control-f and logon and specify the proper termtype in your HELLO command. No ENQ is output by the ADCC and ATP upon receipt of the initial carriage return from the terminal. Thus, you do not get hung immediately. You must still specify the proper termytype in your HELLO command to avoid getting hung on an ENQ output later.

The XON/XOFF flow control protocol is controlled by the terminal. When the terminal wishes to suspend output from the HP-3000 it sends an XOFF (control-s or DC3) to the HP-3000 and sends an XON (control-q or DC1) to resume output. Unfortunately the HP-3000 sometimes fails to properly handle one of the two characters and you either overflow your terminal or get hung up. This is particularly nasty when your terminal is a receive-only printer and you can't supply a missing XON. You're really dead if the HP-3000 misses the XOFF. XON/XOFF is not handled well by the ADCC and ATP. Neither controller strips the parity bit of incoming characters when pass-through parity is in effect. Thus, a terminal using any parity other than space will have all of its XONs or XOFFs ignored; the character with the parity bit included will not match the character used by the controller because its parity bit is set to zero.

A special note on XON. If you inadvertently send an XON (DC1) to the HP-3000 (ATC only) when output is not suspended, you will be in paper tape mode and backspace, control-x, and linefeed will act strangely. Hit a single control-y to get out of this mode.

Some terminals perform flow control by raising and lowering a signal on their interface, the HP-3000 can not handle this. You must either run the terminal at a low enough speed to avoid overflowing it or provide hardware to convert the high/low signal to ENQ/ACK or XON/XOFF.

The form of flow control used by HP terminals when block mode is enabled is the DC1/DC2/DC1 protocol. When the enter key is pressed on the terminal, a DC2 is sent to the HP-3000 after receipt of a DC1 to alert the HP-3000 of a pending block mode transfer. When the HP-3000 is ready to receive the data it sends a DC1 back to the terminal to start the data transfer. (Your program does not handle the DC2/DC1, but see below FCONTROL 28, 29.)

This works fine except in certain circumstances. In certain modes the terminal actually sends DC2 carriage return when the enter key is pressed. This is no problem unless the DC2 and CR do not arrive at the HP-3000 together. The CR may be seen as the end of the data if it comes sufficiently far behind the DC2, your program completes its request for data with nothing and the real data bites the dust when it finally shows up. The separation of the DC2 and CR can occur when using statistical multiplexors or when using Telenet or Tymnet. Be aware, this problem is infrequent, but unsettling when it occurs. The ADCC and ATP attempt to solve this problem by deleting any carriage return that follows a DC2, regardless of the distance between them.

Special Considerations

Every shop should have the proper tools to perform its tasks. Don't neglect your terminal needs. Keep on hand a small flat blade screwdriver, a small Phillips head screwdriver, needlenose pliers, and some sort of breakout box. A breakout box is a small box which is placed inline between two devices. It allows you to monitor, via LEDs, certain RS-232 leads. This tool lets you visually verify the state of modem signals and data flow. It allows recabling by providing a jumper area so that any pin to pin combination desired is achievable. These boxes cost from $38 to $200.

The RS-232 specification requires that terminals running at high speed be no further than 50 feet apart. Almost everyone ignores this specification and experiences no problems. ATP users are beginning to have the rule enforced, the ATP seems more sensitive to long RS-232 lines. Asynchronous line drivers (short haul modems) can be used to drive terminals at greater distances. The HP implementation of RS-422 specifies a maximum separation of 4000 feet at high speeds. This improvement, as well as the fact that the interface is almost immune to noise, will simplify certain aspects of terminal interconnection in the future. Unfortunately, the two specifications are incompatible and switching from RS-232 to RS-422 is not trivial.

A multiplexor is a device that allows many terminals at a remote location to be connected to the HP-3000 over a single communication line. Each terminal is connected to a distinct port on the HP-3000, but savings are realized because all terminals share the same phone line. Multiplexors attempt to maximize the shared use of the line, but in doing so have to use flow control protocols if the aggregate data rate from the terminals or computer exceeds that of the data communicaton line. Multiplexors can be set up to use XON/XOFF flow control protocol at either end. On the computer end this usually causes little trouble. On the terminal end, though, problems abound. Neither the user, typing in character mode, or the terminal, transmitting in block mode, is likely to obey an XOFF. Many block mode terminals sharing a line may overflow it if sufficient capacity is not available.

The ENQ/ACK protocol does not perform well with some multiplexors. The delay between the transmission of the ENQ from the HP-3000 and the subsequent receipt of the ACK generated by the terminal can cause the terminal to print in a start/stop mode. Some multiplexors attempt to solve this problem by emulating the ENQ/ACK protocol at each end. The multiplexor at the HP-3000 end responds to ENQ with an ACK and passes the ENQ to the remote multiplexor. The remote multiplexor passes the ENQ to the terminal and waits for the ACK response. This feature allows the transmission to proceed much more smoothly.

The value added networks, Tymnet and Telenet, also use XON/XOFF and have the same problem with sending an XOFF to the terminal as do multiplexors. The networks can be configured not to use XON/XOFF, but you can still lose data. There is a new termtype, 24, that is used for virtual terminal ports connected through an INP. The INP, Tymnet/Telenet, and certain HP terminals can work together to make block mode work effectively without data loss.

A port selector is a device that allows many terminals to be connected to not-so-many computer ports. Terminals are assigned to ports on a first-come, first-served basis until all ports are consumed. Subsequent terminal service requests are refused or held until a port becomes available. Beyond this basic operation, port selectors can implement priority schemes, allocate terminals to ports on different computers, and perform automatic disconnects. One question arises though. How does the port selector know when a port on the HP-3000 is available? When a terminal logs off a direct connect port, subtype 0 or 4, there is no indication other than the logoff message that the port is free. For ports controlled by port selectors, use of subtypes 1 or 5 causes a modem signal to drop on logoff. This can be used by the port selector as an indiction that the port is free. Some systems require that all terminals provide a terminal ready modem signal. When the terminal is powered off, the signal is lowered and the port to which the terminal was connected is considered free. Other systems require that the user type a disconnect sequence whenever they cease using a port.

There is one very large potential problem with port selectors that must be dealt with carefully. Suppose there is a brief power outage in the computer room; the HP-3000 powerfails and maintains all sessions intact. The port selector will probably lose all connections requiring that the users re-establish their connections. However, there is no guarantee that each user will get the same port, and since all sessions are still active, everyone winds up in someone else's session. (Something to think about before your next security audit). You will have to manually abort all sessions on the HP before allowing the users to reconnect through the port selector.

PROGRAMMATIC DEVICES

Attaching a terminal as a programmatic device is usually done when you want to attach a serial printer, instrument, data collection device, or other strange beast to the HP-3000. An application program you write will typically control all access to the device; a user will not walk up to it, hit return, and log on. I will explain the various intrinsics that are used to access programmatic devices.

In the following intrinsic descriptions, differences between ATC, ADCC and ATP machinces will be noted. However, it is finally the case that the ADCC and ATP behave identically except in the case of parity. The ATC is essentially frozen and does differ significantly in some areas.

FOPEN

You must call FOPEN to gain access to the device. I always use a formal file name to allow control of the open with file equations. If the device is unique in the system, I use its device name as the file name. The foptions specify CCTL, undefined length records, ASCII, and a new file. The aoptions specify nobuf, exclusive access and input/output. Choose a record size that is larger than the maximum data transfer that will take place.

For devices that are to be used exclusively in programmatic mode it is recommended that you REFUSE the device so that extraneous carriage returns from the device will not be interpreted as logon attempts by the HP-3000.

Opening subtype 1 or 5 ports differs among the controllers regarding the point at which you hang if the controller finds that the modem is not online. Programatically handling incoming calls can be tricky; experiment carefully.

FCLOSE

You call FCLOSE to release access to the device. Though FCLOSE resets most FCONTROL options, it is good practice to explicitly reset all FCONTROL options before calling FCLOSE.

ATC - MPE sends a CR/LF to the device if it believes that the "carriage" is not at the beginning of the line, i.e., the last character output was not a linefeed.

ADCC/ATP - MPE never sends a CR/LF.

FREAD

You call FREAD to get data from the device. Many of the FCONTROL calls shown below affect how FREAD works. End-of-file is indicated by a record that contains ":EOF:". Any record with a colon in column one is an end-of-file to $STDIN. ":EOD", ":EOJ", ":JOB", ":DATA", and ":EOF:" are end-of-file to $STDINX.

The default end of record terminator is carriage return. You should change this if the device terminates all records with some other character. You can specify an alternate terminator that will terminate a record in addition to carriage return. Choose the teminator so that it is the last character input. For a device that sends a linefeed after carriage return, try to use linefeed as the terminator instead of carriage return. See FCONTROL 41 below.

Some devices send data followed by a fixed terminator followed by a LRC or CRC character. This error checking character can take on all values, thus it can not be used as a terminator. Unfortunately, it often looks like XOFF which will halt any further output to the device.

You may want to trap certain errors returned by FREAD to your program: 22, software time-out; 31, end of line (alternate terminator); and 28, timing error or data overrun. This last error occurs frequently on ADCCs running at high speeds.

ATC - The characters NULL, BS, LF, CR, DC1, DC3, CAN (control-x), EM (control-y), ESC:, ESC;, and DEL are stripped from the input stream for both session and programmatic devices.

ADCC/ATP - The characters NULL, BS, LF, CR, DC1, DC2, DC3, CAN (control-x), EM (control-y) and DEL are stripped from the input stream for both session and programmatic devices. There is a patch available that deletes recognition of ESC: and ESC; while solving certain overrun problems, the patch is strongly recommended. A DC2 at the beginning of the line causes the driver to send another DC1 to the terminal, as it thinks that a block mode read has started.

Each time you issue an FREAD to the terminal, MPE sends a DC1 to the terminal to indicate that it is ready to accept data. Most devices ignore, totally, the DC1. If your device reacts negatively to the DC1, use termtype 18 which suppresses the DC1 on terminal reads. The device must not send data to the HP-3000 until it has received the DC1, otherwise the data will be lost. If the device does not wait for the DC1 you must supply external hardware that will provide buffering and wait for the DC1; or you can solve the problem on the HP-3000 by using two ports to access the device. One port is opened for reading and the other for writing. A no-wait read is issued before the write that causes the device to send data, then the read is completed. Connect the terminal's pin 2 (Transmit Data), to the read port's pin 2; connect the terminal's pin 3 (Receive Data), to the write port's pin 3; and connect the terminal's pin 7 (Signal Ground), to pin 7 of both ports. (This two port scheme was first introduced to me by Jack Armstrong and Martin Gorfinkel of LARC.)

The ADCC and ATP now offer another possible solution to this buffering problem. If the device waits for some character from the computer before transmitting data, the FDEVICECONTROL intrinsic can be used to have the driver send that character rather than DC1 on each read. Your program would write the prompt sequence less the final trigger character, then perform the read which supplies the trigger character.

FWRITE

You call FWRITE to send data to the device. The carriage control (cctl) value of %320 is often used to designate that MPE send no carriage control bytes, such as CR/LF, to the device. Control returns to your program from FWRITE as soon as the data is loaded into the terminal buffers, MPE does not wait until all data has been output to the device. If you must know when the actual output is complete, the temptation is to use FSETMODE to enable critical output verfication. Unfortunately, this isn't implemented for terminals. However, you can achieve the same effect by calling FCONTROL to set the desired echo state after each write. The FCONTROL will not complete until the write is physically complete.

ATC - the initial FWRITE to an HP terminal termtype causes an ENQ to be sent to the terminal.

ADCC/ATP - the initial FWRITE does not send an ENQ.

FSETMODE - 4 - Suppress carriage return/linefeed

In normal operation a line feed is sent to the terminal if the input line terminates with a carriage return, a CR/LF is sent to the terminal if the line terminates by count, and nothing is sent if the line terminates with an alternate terminator. These extra characters may not be desirable in certain applications. FSETMODE 4 suppresses these linefeeds and carriage returns. FSETMODE 0 returns to normal line termination handling, an FCLOSE also returns the device to the normal mode.

FCONTROL

FCONTROL is the workhorse intrinsic for managing a programmatic device on the HP-3000. Each use of FCONTROL will be shown separately but it will usually be the case that several calls will be used. Most calls are required only once, but the timer calls are required for each input operation. Each call will be identified by the controlcode parameter that is passed to FCONTROL.

FCONTROL - 4 - Set input time-out

This option sets a time limit on the next read from the terminal. It should always be used with devices that operate without an attached user to prevent a "hang." If something goes wrong with the device, your program will not wait forever; control will be returned eventually. The FREAD will fail and a call to FCHECK will return the errorcode 22 (software time-out). No data is returned to your buffer in the case of a time-out; any data entered before the time-out is lost.

If you issue a timeout for a block mode read on the ATC, the timer is stopped if a DC2 is received from the terminal. A new timer is started which runs for 30 seconds plus the expected data transfer time. If the read doesn't complete, an error 27 is reported. On the ADCC/ATP, receipt of the DC2 dces not stop the timer. It continues to run and if the read doesn't complete, an error 27 is reported. In all cases, an error 22 is reported if no DC2 is received and the read doesn't complete. To get the ADCC/ATP to start a new timer on receipt of a DC2, FCONTROL 31 must be used which starts a new timer for 10 seconds plus the expected data transfer time.

FCONTROL - 10, 11 - Set terminal input/output speed

These FCONTROL options allow you to change the terminal input and output speeds. FCONTROL 37 can also be used to set terminal speed. It sets termtype as well and is the method that I prefer.

ATC - Split speeds are allowed.

ADCC/ATP - Split speeds are not allowed, FCONTROL 10 does nothing and FCONTROL 11 sets both input and output speed.

FCONTROL - 12, 13 - Enable/disable input echo

These FCONTROL options allow you to enable and disable terminal input echoing. Many devices that attach to the HP-3000 do not expect or desire echoing of the characters they transmit. This option, along with FSETMODE 4, completely turns off input echoing. Echoing is not restored when a file is closed, so you should always put echo back the way it was found.

FCONTROL - 14, 15 - Disable/enable system break

The break key should be disabled if terrible things happen when the user hits break and aborts out of a program. You, the programmer, always seem to need break for debugging purposes and discover that you have it turned off. System break can only be enabled for session devices, it is not allowed for programmatic devices. If break is entered on a session device, the data already input will be retained and provided to the user program after a resume and completion of the read. If a break is entered on a programmatic device, a null will be echoed to the device, but no data is lost.

FCONTROL - 16, 17 - Disable/enable subsystem break

Subsystem break is recognized only on session devices; it can be enabled on programmatic devices but has no effect. If a control-y is entered during a read, the read terminates and the data already input will be retained and provided to the user program after the control-y trap procedure returns. If control-y is disabled, any control-y will be stripped from the input but no trap procedure is called and the read continues. Control-y trap procedures are armed by the XCONTRAP intrinsic. A subsystem break character other than control-y may be specified when unedited terminal mode (FCONTROL 41) is used. In programmatic mode, the subsystem break character is always stripped from the input stream.

FCONTROL - 18, 19 - Disable/enable tape mode

ATC - This is effectively an FSETMODE 4, an FCONTROL 35, and suppression of backspace echoing, all rolled into one.

ADCC/ATP - Tape mode can not be enabled.

FCONTROL - 20, 21, 22 - Disable/enable terminal input timer, read timer

These options can be used to determine the length of time it took to satisfy a terminal read. It is not a time-out, that is FCONTROL 4. The manual states that you must enable the timer before each read, so why is there a disable option? If you read the timer without enabling the timer, you get the time of the most recent read that did have the timer enabled. The number returned is the length of the read in one-hundreths of a second.

FCONTROL - 23, 24 - Disable/enable parity checking

This option enables parity checking on input for the parity sense specified by FCONTROL 36.

ATC - This option affects input parity checking only, output parity generation is controlled by FCONTROL 36.

ADCC/ATP - This options controls both input parity checking and output parity generation, FCONTROL 36 only specifies the type of parity.

FCONTROL - 25 - Define alternate line terminator

This option is used to select an alternate character that will terminate terminal input in addition to carriage return. It is useful if your device terminates input with something other than return. No CR/LF is echoed at line termination.

ATC - NULL, BS, LF, CR, DC1, DC3, CAN, EM and DEL are not allowed as terminators. The manual claims that DC2 and ESC are not allowed as terminators, but they work. If a DC2 is the first input character from an HP termtype terminal, the HP-3000 drops the DC2 and sends a DC1 back to the terminal, and thinks a block mode transfer is starting. Any other DC2 is recognized as a terminator, if enabled. By enabling user block mode transfers (FCONTROL 29), a DC2 as the first character will also be recognized as a terminator when enabled. For non-HP termtype terminals a DC2 is always recognized as a terminator when enabled.

ADCC/ATP - everything (except NULL) is allowed as an alternate terminator, even carriage return.

If a line terminates with the alternate terminator, the character will be included in the input buffer and counted in the length. A read terminated by the alternate character always returns an error condition. You must call FCHECK to determine that the read terminated with the alternate character, which is indicated by errorcode 31.

FCONTROL - 26, 27 - Disable/enable binary transfers

Binary transfers can be used to transmit full 8-bit characters to and from the terminal. On input, a read will only be satisfied by receiving all characters requested, a carriage return (or alternate terminator) will not terminate the read. Thus, you must always know how many characters to read on each input from the terminal. Enabling binary transfers also turns off the ENQ/ACK flow control protocol and carriage control on output. No special characters are recognized on input. No CR/LF is echoed to the terminal at the end of the read. If a session device is being accessed in binary mode, a break will remove the terminal from binary mode but it will not be returned to binary mode when a resume is executed.

FCONTROL - 28, 29 - Disable/enable user block mode transfers

As described above, the normal sequence of events in a block mode transfer from an HP terminal to the HP-3000 is for the HP-3000 to send a DC1 to the terminal indicating its readiness to accept data. The terminal sends a DC2 when the enter key is struck to indicate that it is ready to send data. The HP-3000 responds with another DC1 when it is really ready to take the data. Finally, the terminal sends the data. All of this is transparent to your program which just issues a big read. If you would like to participate in this handshake you enable user block mode transfers and MPE relinquishes control of the handshake. Your program would issue a small read, get the DC2, and issue another read to accept the data. This allows you to meddle before the data shows up.

FCONTROL - 30, 31 - Disable/enable V/3000 driver control

This option is an undocumented option in which the terminal driver provides low level support for V/3000 use of terminals. When V/3000 issues a read to the terminal, the driver outputs a DC1; the terminal user hits enter, which causes a DC2 to be sent to the 3000; the driver responds with ESC H ESC c DC1, which locks the keyboard and homes the cursor; it appears that the driver also enables binary transfers, because the second read only terminates by count, not by terminator. Until the DC2 is received, the read looks like an unedited mode read with CR as the terminator, except that the read doesn't fail. Any characters received before a DC2 are discarded (as are an indeterminate number of characters that immediately follow the DC2). Typically, this would be a CR(LF) when the terminal is operating in block, line mode. The ATC sends ESC c ESC H (just a little inconsistency to keep you awake.)

The terminal driver only supports block mode transfers with HP termtypes and performs one other function during block mode transfers. Normally you wouldn't put a timeout (FCONTROL 4) on a block mode read because the user can take an indefinite amount of time to fill a screen; but you would like to avoid terminal hangs because data or the terminator from the terminal gets lost. This situation is handled by the driver for you; the portion of the read after receipt of the DC2 is timed for (#chars in read/#chars per sec)+10 seconds. If some data or the terminator is lost and the read times out, the read will fail and FCHECK will return error 27. The ADCC/ATP do not perform this function unless FCONTROL 31 is in effect. The ATC performs this function regardless of the FCONTROL 31 state and uses 30 seconds instead of 10 seconds when starting the timer.

FCONTROL - 34, 35 - Disable/enable line deletion echo suppression

Option 35 disables the !!! CR/LF echo whenever a control-x is received from the terminal. The control-x still causes all data to be deleted from the input buffer. The ADCC/ATP disables the !!!, but not the CR/LF.

FCONTROL - 36 - Set parity

This FCONTROL option sets the sense of the parity generated on output and checked on input. The four possibilities are: 0, space or no parity, all 8 bits of the data are passed thru; 1, mark parity, the parity bit is always set to one; 2, even, even parity is generated on all characters; and 3, odd parity, odd parity is generated on all characters.

An undocumented effect of this FCONTROL call is that the previous parity setting is returned in the "param" parameter, wiping out its original value!

ATC - FCONTROL 36 sets the parity sense and enables output parity generation. FCONTROL 24 must be called to enable parity checking on input.

ADCC/ATP - FCONTROL 36 sets the parity sense only. FCONTROL 24 must be called to enable output parity generation which results in input parity checking, as well.

On the ATC, parity is not reset to the default when a device is closed. This can be useful if you have a session device that can not run with the default parity. Each time the system is started, run a program that: opens the device, sets the parity, and closes the device. It can then be accessed as a session device with the required parity.

On the ATC, the default parity for programmatic devices is odd parity on output and no parity checking on input with the parity bits set to zero. The ADCC and ATP default to parity pass-through for programmatic devices. Note that there is no way on the ADCC/ATP to request the default parity setting. This makes it difficult to write a program that sets parity pass-through without knowledge of the machine type; except for a subtle trick: issue an FCONTROL 36/0 without an FCONTROL 24. On the ATC, this results in parity pass-through on output, with parity stripping (without checking) on input. Without an FCONTROL 24, the FCONTROL 36 has no effect on the ADCC/ATP, leaving the default parity pass-through in effect.

The following tables shows the results of testing the various parity options. In each case, both FCONTROL 24 and FCONTROL 36 were specified so that parity generation was enabled on output and parity checking was enabled on input.

Option 0 - Space parity or parity pass-through

ATC - pass-through parity on output, did no checking and stripped parity bits on input.

ADCC - generated even parity on output, checked for even parity on input.

ATP - generated space parity on output, checked for even parity on input.

Option 1 - Mark parity

ATC - generated mark parity on output, did no checking and stripped parity bits on input.

ADCC - generated odd parity on output, checked for odd parity on input.

ATP - generated mark parity on output, checked for odd parity on input.

Option 2 - Even parity

ATC/ADCC/ATP - generated even parity on output, checked for even parity on input and stripped parity bits.

Option 3 - Odd parity

ATC/ADCC/ATP - generated odd parity on output, checked for odd parity on input and stripped parity bits.

FCONTROL - 37 - Allocate a terminal

In the old days you had to allocate a programmatic terminal before it could be used - now you don't. This option is still useful, however, because you can set the termtype and terminal speed with one FCONTROL call. Common sense (mine at least) says to set termtype and speed each time a device is opened, even if the proper values are configured.

FCONTROL - 38 - Set terminal type

This option allows you to set the terminal type, but use FCONTROL 37, and set type and speed all in one shot.

FCONTROL - 39 - Obtain terminal type information

Before changing the terminal type, get the current value and reset it when you are through.

FCONTROL - 40 - Obtain terminal output speed

Before changing the terminal speed, get the current value and reset it when you are through.

FCONTROL - 41 - Set unedited terminal mode

Unedited terminal mode is probably the most useful FCONTROL option used to communicate with programmatic devices. It allows almost all control characters to pass through to the HP-3000 without requiring reads of exact length, as in binary transfers. Input will terminate on a carriage return or an alternate terminator, if specified. The subsystem break character, replacing control-y, can also be specified, but is only effective on session devices.

ATC - all input parity bits are set to zero (7 bit mode). Unedited mode overrides parity checking.

ADCC/ATP - all input parity bits are passed through (8 bit mode). Parity checking, when enabled, overrides unedited mode and all input parity bits are set to zero (7 bit mode).

Binary transfers, when enabled, override unedited terminal mode enabled. If the input terminates with the end-of-record character or alternate terminator, no CR/LF is sent to the terminal. If the input terminates by count, a CR/LF is sent to the terminal unless an FSETMODE 4 has been done. Unedited mode does not turn off the ENQ/ACK flow control protocol.

FDEVICECONTROL

This intrinsic is related to FCONTROL in the same mánner that FFILEINFO is related to FGETINFO. Many of the same functions are supported, as are many new functions. FDEVICECONTROL provides intrinsic access to the new features supported by the Workstation Configurator. Unfortunately, the manual states that many of the features will not be supported at all in the future or will not be supported programmatically. Read the manual carefully before using these virtual features.

PTAPE

The manual describes PTAPE as the intrinsic to use to read paper tapes. (Paper tape is a fancy data-entry media that is becoming increasingly popular.) It can be used on the HP-3000 to access devices that send up to 32767 characters all in one shot, subject to a few limitations. The data must be record oriented with carriage returns between records; MPE will cut the data into 256 character records if there are no returns; and the whole mess must be terminated by a control-y. Certain buffering terminals allow you to: fill their memory off-line, connect to a computer, and transmit all the data. This could save considerable time and money over dial-up phone lines.

DEBUGGING

If you have a requirement to attach a programmatic device to the HP-3000, the worst strategy is to write some code on the HP-3000, plug the device in and start testing. Murphy says it won't work and it won't. The method I use is to test the device, then test the code, and then test the code and the device together. I test the device by plugging it into an HP-2645 (or equivalent) terminal, turning on monitor mode, and simulate the HP-3000 by typing on the keyboard. (Remember that you are hooking two terminals together; you will probably hook device pin 2 to 2645 pin 3, device pin 3 to 2645 pin 2, and device pin 7 to 2645 pin 7.) You can stimulate the device and observe all responses quite simply. Any strange behavior can be noted at this point. The next step is to write the code on the HP-3000 to access the device in the manner determined by the first tests. Then plug the HP-2645, not the device, into the HP-3000. Now type on the 2645 to simulate the device, continue until your code is debugged. Now you can plug the device into the HP-3000 and you have a good (modulo Murphy) chance of actually getting it to work.

REFERENCES

Data Communications Handbook (Fundamental), Hewlett-Packard Company, June 1984, Part # 5957-4634.

Roseville Terminals Cabling Manual, Hewlett-Packard Company, Part # 5957-9918.

Black Box Catalog, P.O. Box 12800, Pittsburgh, PA 15241, (412) 746-5500 A catalog that is required in every shop.

HP Point-to-Point Workstation I/O Reference Manual, Hewlett-Packard Company, December 1984, Part # 30000-90250.

3057. PERFORMANCE OPTIMIZATION IN COBOL

Bruce Tobak
2205 Fulton Road
La Verne, California 91750

The first standard COBOL was proposed in 1960, when computers were very different from today's mini- and microcomputers. They were single-user, multi-priest behemoths with architectures designed to fit the electronic components merely expensive. The processor itself required a large corporation and a large staff to purchase and maintain it.

The original COBOL standard was designed with these facts in mind, and many of the programming practices and accumulated wit and wisdom of the COBOL programmer come from those times. Presented here is a selection of that wit and wisdom, and how it relates to COBOL/3000 (and indeed most modern computers).

1. Indexing is faster than subscripting.

Remember registers? (If you ever programmed in assembly language, you certainly do. If you've only programmed in a high-level language, you probably don't.) General purpose computers usually had a limited number of these high-speed memory locations, generally eight or sixteen. The compiler, when generating code for a COBOL source program used several of these for itself, but the remainder were available to the programmer for USAGE IS INDEX items. Indexing was much faster than subscripting because in order to access an array element, a subscript would have to be brought into a register, and possibly converted to a different data type. Indexing, by definition, meant using an item which was already in a register, so bypassing the conversation and data movement steps. Worse still, some computers had addresses that could be operated on only by special instructions (e.g., the Burroughs B200/300/500 series). This is the reason that indices can only be added to or subtracted from.

On the HP3000, though, there is only one index register, andit is shared by all arrays. Therefore, as long as the subscript you are using is USAGE COMP PICS9(4), there is no difference between indexing and subscripting - either in the generated code or the speed of the resulting program. Shops which try to speed up programs by converting them to use indexing would be better off devoting the time yo programmers' vacations: system performancewould then at least be improved because of a smaller program development load! However, performance can be improved by changing your subscripts from COMP-3 to COMP: the compiler emits code necessary to do the required conversion, but this is relatively expensive in run time. (But see 5.)

2. COBOL sorts take longer than external sorts.

This is very application dependent, and again has strong roots in history and folklore. In the Dark Times, the COBOL SORT verb generated in-line code to call some routines the compiler folks wrote. The compiler folks generally had better things to do than write sorts, so the sorts were not necessarily very good. In addition, these sorts were not very adaptive to circumstances, and had a limited performace range. They might involve additional overlays that had to be read in from (heaven forbid!) cards. Besides, in a batch-oriented, job-step environment there was not much point to an internal sort. On the HP3000, though, all sorting is done by the SORT/3000 subsystem, regardless of whether you use SORT.PUB.SYS or the COBOL SORT verb: the HP3000 COBOL compilers simply emit code to call SORT/3000 intrinsics on behalf of your program. The result of this is that sorts done with the same sets of keys and in the same kinds of environments will take the same amount of time, regardless of whether you envoke them through COBOL or through MPE.

The key here is in the same kinds of environments. SORT/3000 needs memory to work: the more, the better. If your COBOL program requires 20k words of memory for itself when you execute the SORT verb, SORT/3000 will get only about 9k. Experiments show that when sorting 80-byte records, SORT/3000 needs about 8k words to produce acceptable performance, and works best with at least 16k words. So, you'll have better performance with an external sort...

Sometimes, to determine whether to use an internal sort or an external one, you should look at your entire application. Many batch-oriented systems converted to run on the HP3000 have jobsteps that include a sort to a temporary file, a report on the temp file, a different sort to a temp file, a report on the new temp file, and so on. This means that each record in your master file is being handled three times: once as imput to the sort, once as output to the temp file, and once again as input from the temp file. Replacing the external sort with a SORT verb with an OUTPUT PROCEDURE reduces this to only one: records are read once by SORT/3000 on behalf of your program, and then passed to your program via the output procedure. (Of course, SORT/3000 handles records several times during its execution, but this number is largely irreducible except by providing more memory for the sort.) The results of this kind of redesign can be dramatic: reductions of 2:1 or 3:1 in runtime are possible.

3. COMPUTE is faster/slower than the ADD/SUBTRACT/MULTIPLY/DIVIDE verbs

Like SORT, this depends on how you use the various computational verbs available to you in COBOL. If the object is to perform a complex series of arithemetic operations, using COMPUTE will generally be faster (by a few microseconds). If you are simply adding values to an accumulator, ADD and COMPUTE will generate exactly the same code, and so there will be no performance difference.

4. COBOL is inefficient at arithmetic.

As the previous point shows, how "efficient" a language is at a particular task depends mostly on how you use it. (This generalization applies only to general-purpose languages.) By following a few simple rules, COBOL is as efficient as, say, FORTRAN at arithmetic. In particular, avoid either implicit or explicit type conversions: don't mix COMP-3 and COMP items, and try not to mix COMP PIC S9(1-4) with COMP

PIC S9(5-9). And in addition, avoid arithmetic with USAGE DISPLAY items, since these always require type conversions. (The HP3000 has instructions to operate directly on COMP and COMP-3 items, but lacks arithmetic instructions to operate on DISPLAY types.) For an extended discussion of this, see Jim May's paper Programming for Performance, published in the Proceedings of the 1982 HP3000 IUG Conference, Edinburgh.

5. Searching: IMAGE and internal techniques. Searches, or table lookups, are very common in data processing operations. You probably do them without thinking in most cases, since every IMAGE master lookup (DBGET mode 7, or DBFIND) is really a search operation. If you routinely use IMAGE to do your table lookups for you, you might be surprised at how much time can be saved by using your own code to perform the same operation.

The examples used for this article are derived from a real-life application: printing a purchase order report. Purchase order records were contained in a detail data set containing four items: a part items: a part number, a vendor number, a purchase date, and a quantity.(Other items were left out for the purpose of this example.) Each part number's corresponding manual master record contains a description for it, and each vendor number's corresponding manual master record contains a description for it, and each vendor number's corresponding manual master record contains the vendor's name and address. To test the methods outlined here, I wrote a small COBOL-II program to generate a simple purchase order report, and then modified it to try different search methods.

In the first example, the item being looked up was the vendor's name. The program in Figure 1 is representative of most such programs: a serial read (or perhaps, a sort output procedure) get each detail record, and then a lookup is performed on the associated manual master to get desriptive information. The lookup is entirely contained in the paragraph GET-VENDOR, so that various lookup techniquescan be tried without making significant changes in the rest of the program. (If your programs are written like this, you should be able to simply lift code from the examples, place it in your system, and enjoy the kudos.)

All of the examples, and the performance information derived from them, comes from a 1/2-megabyte Series 30 running T-delta-1 (MPE-V/T) and a single 7925 disc drive. You should keep this in mind when you examine the performance data. If you are running a faster system (you 169 a faster system (you couldn't possibly be running a slower one), you should divide the "CPU-seconds" figures by 2 if you are on a Series III or Series 37; by 5 on a Series 4x CPU; or by 12 (!) if you are on a Series 6x machine. Clock times will not scale by as much, since non-cached I/O rates for a single disc drive are almost identical on all CPU's. In addition, all timing was done with disc caching turned off. Catching will in most cases improve wall-time performance, but leave CPU time almost unchanged. (Disc caching resulted in a slight performance degradation on on this small-memory Series 30.)

The first test was run using the "standard" program in figure 1. To produce this 9,000-line report took just under thirteen minutes, and used 475 CPU seconds. (If you are scaling these numbers for your Series 68 CPU, you should come out with about 5:45 clock time, and 40 CPU seconds.) Is any improvement possible? Since by now you have looked at the graph in figure 5, you know the answer is "yes". The program in Figure 2 replaces the DBGET with a SEARCH verb, which performs a serial search of a table in memory. Of course, some preparation is required for this technique, and this is done in the paragraph INIT-VENDOR-SEARCH. INIT-VENDOR-SEARCH does a simple serial read of VENDOR-MASTER, and saves the vendor number and vendor name in VENDOR-TABLE. The result of running Program 2 is almost a 2:1 reduction in run-time, and savings of about one-third in CPU time. Clearly the extra effort of creating a table in memory was well-spent.

COBOL, though, supports an even faster search verb: SEARCH ALL. Using SEARCH ALL requires that the table used for the search be sorted into ascending or descending order. This, of course, requires some additional preparation, shown in Figure 3. SORT-VENDOR-TABLE is a generalized Shell sort, and you should be able to use this in any of your programs by changing only the identifiers used for the table. For sorting tables that fit entirely in memory, it is much faster than the SORT verb. (The sort would execute faster still if it were programmed in SPL, but the difference for small tables is not worth the the extra effort.)

The result of using SEARCH ALL is a further improvement, although not as dramatic a change as eliminating IMAGE from the picture. The ratio will improve noticeably, though, with larger tables, as you will see shortly.

There is a search method even faster than binary search, however, for most commercial data. The "80-20" rule is a generalization about most things in business: twenty percent of the customers generate eighty percent of the revenue (or orders, or complaints); twenty percent of your vendors are responsible for eighty percent of your shortages, and so on. (As practical examples, think about the number of transactions in your accounting system that use the "accounts payable" or "inventory" accounts.) Because your computer records will (should) reflect the parameters in your business, you will probably discover that eighty percent of the records in a detail are linked to only twenty percent of your master entries. (Of course, for pure "control" information such as invoice numbers, this is not true.) This will probably apply to parts in inventory, to purchase order line items, to sales order items, and other details requiring "descriptive" information from associated masters.

You can use this property of business data to come up with a new search rule for your memory array: Whenever an item is found in the table, it is moved up one entry. Repeated application of this rule rapidly and automatically organizes your table by frequent use. The resulting program is shown in Figure 4; the very simple change is in paragraph GET-VENDOR. About 20 percent less clock time is required now, and about 15 percent less CPU time as well. (The test data used for these timings was designed to have approximately an 80-20 distribution.)

To summarizethe information gathered so far, look at Figure 5. (You've probably already done this, but look again.) Clearly, the "standard" method is the worst of the lot. Given the small effort involved in adding any of these internal searches, they are probably an easy way to gain performance. The small amount of "overhead" time in each bar is the time required to do any pre-processing before the report starts. In program 1, the time is spent only in opening the data base and output files. In programs 2 and 4, there is additional time required to read the master data set, and in program 3, some additional time is required to sort the array. The overhead time is very low for all three "fast" methods.

These techniques will work on small master data sets (a few hundred entries or so), but what about masters too large to fit into the stack? (Unless you are sorting using a seperate process, you should leave at least 12K words for any required sort.) In the case of the vendor file, the maximum would be about 1100 entries. In the case of the part master, however, the maximum would be only about 650 entries. (In a practical program, there would be even fewer entries, since these small test programs make no allowance for V/3000 space, extra files open, or other uses of the stack. Moreover, there would probably be several manual masters on which lookups were to be performed.) A modification of the technique used in Program 4 can still be used as long as the master data set or sets are not extremely large. In this case, rather than reading the entire master data set into memory at once, entries are read in "on demand." No initial "preload" of the array is performed. Instead, the table size is set to zero during initialization, and as each detail record is read, a table search is performed. If the table search fails, the program looks up the key in the appropriate master data set, and the newly read record is added to the growing array. (If the requested record is found in the array, its position is adjusted as in Program 4.) When the array becomes full, there are two choices. First, the entry at the bottom of the array can be thrown away to make room for the new entry. This method will efficiently handle cases in which detail records with a particular key are "clustered" - newly-read records will tend to migrate to the top of the array as their "cluster" is read.

6. A perspective on performance

With the exception of searching and sorting, most of the performance questions analyzed here, and most performance debates in general are matters of microseconds on modern computers. You should take this into account before embarking on any large rewrite projects. For example, changing a subscript from COMP-3 to COMP may save 50 microseconds (millionths of a second) per array access. If you access the table in question ten million times over the course of a three-hour run, your rewrite will save 500 seconds, or about eight minutes out of 180 - a 4-1/2 percent improvement.

Conversely, if you are doing an external sort followed by a report, for a total run-time of three hours, changing the application to run with a single internal sort might save an hour or more, a savings which would justify a two-day redesign. (Assuming that the report is run more often than once a year.)

Finally, performance is much more a matter of choosing the right design in the first place, rather than "tweaking" a poor design to get more speed out of it. As the sorting example shows, shaving 20 percent off the run-time of a bad design will usually pay a lot less than finding a good design to start with. For an excellent discussion of this, see The Elements of Programming Style, Kernighan and Plauger (1974), Chapter 6.

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.12  COBOL II/3000 FRI, AUG  2, 19

001000$CONTROL USLINIT, LINES = 56, MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. TESTSRCH.
001300 ENVIRONMENT DIVISION.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600     SELECT REPORT-OUTPUT ASSIGN TO "LPFILE,UR".
001700 DATA DIVISION.
001800 FILE SECTION.
001900 FD  REPORT-OUTPUT LABEL RECORDS ARE OMITTED.
002000 01  OUTPUT-AREA PIC X(80).
002100 WORKING-STORAGE SECTION.
002200 01  OUTPUT-REC.
002300     05  PART-NUMBER   PIC X(16).
002400     05  FILLER        PIC X(2) VALUE SPACES.
002500     05  VENDOR-NUMBER PIC Z(4).
002600     05  FILLER        PIC X(2) VALUE SPACES.
002700     05  PURCHASE-DATE PIC X(6).
002800     05  FILLER        PIC X(2) VALUE SPACES.
002900     05  QUANTITY-PURCHASED PIC Z(7)9.99- USAGE DISPLAY.
003000 01  DBSTAT.
003100     05  DB-STATUS PIC S9(4) USAGE COMP.
003200     05  DBSTAT-ELEMENT PIC S9(4) USAGE COMP OCCURS 9.
003300 01 VENDOR-REC.
003400     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
003500     05  VENDOR-NAME   PIC X(20).
003600 01  PURCHASE-REC.
003700     05  PART-NUMBER   PIC X(16).
003800     05  VENDOR-NUMBER PIC S9(4) COMP.
003900     05  PURCHASE-DATE PIC X(6).
004000     05  QUANTITY-PURCHASED PIC S9(8) COMP.
004100 77  DATA-BASE PIC X(16) VALUE "  PURCH  ".
004200 77  VENDOR-MASTER PIC X(16) VALUE "VENDOR-MASTER   ".
004300 77  PURCHASE-FILE PIC X(16) VALUE "PURCHASE-FILE   ".
004400 77  PART-MASTER   PIC X(16) VALUE "PART-MASTER     ".
004500 77  DSET-NAME     PIC X(16).
004600 77  PASSWORD      PIC X(16).
004700 77  IMAGE-LIST    PIC X(32).
004800 77  SAME-LIST     PIC X(2) VALUE "*;".
004900 77  ALL-ITEMS     PIC X(2) VALUE "@;".
005000 01  DUMMY.
005100     05 FILLER    PIC S9(4) USAGE COMP.
005200 77  MODE-1        PIC S9(4) USAGE COMP VALUE 1.
005300 77  MODE-2        PIC S9(4) USAGE COMP VALUE 2.
005400 77  MODE-3        PIC S9(4) USAGE COMP VALUE 3.
005500 77  MODE-4        PIC S9(4) USAGE COMP VALUE 4.
005600 77  MODE-5        PIC S9(4) USAGE COMP VALUE 5.
005700 77  MODE-6        PIC S9(4) USAGE COMP VALUE 6.
005800 77  MODE-7        PIC S9(4) USAGE COMP VALUE 7.
005900 77  DB-MODE       PIC S9(4) USAGE COMP.
006000 77  NAME-INDEX    PIC S9(4) USAGE COMP.
```

```
PAGE 0002/COBTEXT  TESTSRCH
006200 PROCEDURE DIVISION.
006300 INITIALIZATION SECTION.
006400 INIT-DB.
006500     CALL "PRINTIME".
006600     MOVE "; " TO PASSWORD.
006700     CALL INTRINSIC "DBOPEN" USING DATA-BASE, PASSWORD, MO
006800      DBSTAT.
006900     PERFORM DB-CHECK.
007000     MOVE "VENDOR-MASTER " TO DSET-NAME.
007100     MOVE 201 TO DB-MODE.
007200     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
007300      DBSTAT, VENDOR-MASTER.
007400     PERFORM DB-CHECK.
007500*
007600     MOVE "PURCHASE-FILE " TO DSET-NAME.
007700     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
007800      DBSTAT, PURCHASE-FILE.
007900     PERFORM DB-CHECK.
008000*
008100     MOVE "PART-MASTER " TO DSET-NAME.
008200     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
008300      DBSTAT, PART-MASTER.
008400*
008500* Set up list for PURCHASE-FILE.
008600*
008700     CALL INTRINSIC "DBGET" USING DATA-BASE, PURCHASE-FILE
008800      MODE-1, DBSTAT, ALL-ITEMS, PURCHASE-REC,
008900      DUMMY.
009000     IF DB-STATUS IS NOT = 17 THEN PERFORM DB-CHECK.
009100*
009200* Set up list for VENDOR-MASTER
009300*
009400     MOVE "VENDOR-NUMBER,VENDOR-NAME;" TO IMAGE-LIST.
009500    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
009600    DBSTAT, IMAGE-LIST, PURCHASE-REC, DUMMY.
009700     IF DB-STATUS IS NOT = 17 PERFORM DB-CHECK.
009800*
009900* Open report file
010000*
010100     OPEN OUTPUT REPORT-OUTPUT.
010200      MOVE ALL SPACES to OUTPUT-AREA.
010300     CALL "PRINTIME".
010400*
010500* End of initialization.
010600*
```

```
PAGE 0003/COBTEXT  TESTSRCH
010800 REPORT-GENERATION SECTION.
010900 GET-NEXT-RECORD.
011000      CALL "DBGET" USING DATA-BASE, PURCHASE-FILE, MODE-2,
011100      SAME-LIST, PURCHASE-REC, DUMMY.
011200      IF DB-STATUS = 11 GO TO CLEANUP.
011300      PERFORM DB-CHECK.
011400      PERFORM GET-VENDOR.
011500      PERFORM PRINT-LINE.
011600      GO TO GET-NEXT-RECORD.
011700*
011800 PRINT-LINE.
011900      MOVE CORRESPONDING PURCHASE-REC TO OUTPUT-REC.
012000      MOVE CORRESPONDING VENDOR-REC TO OUTPUT-REC.
012100      MOVE OUTPUT-REC TO OUTPUT-AREA.
012200      WRITE OUTPUT-AREA.
012300*
012400 GET-VENDOR.
012500      CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTE
012600         MODE-7, DBSTAT, SAME-LIST, VENDOR-REC,
012700         VENDOR-NUMBER OF PURCHASE-REC.
012800      IF DB-STATUS IS = 17
012900         DISPLAY "Expected vendor not found - ",
013000           VENDOR-NUMBER OF PURCHASE-REC
013100         GO TO CLEANUP.
013200      PERFORM DB-CHECK.
013300*
013400 CLEANUP.
013500      CALL INTRINSIC "DBCLOSE" USING DATA-BASE, DSET-NAME,
013600       MODE-1, DBSTAT.
013700      CLOSE REPORT-OUTPUT.
013800      CALL "PRINTIME".
013900      STOP RUN.
014000*
014100 DB-CHECK.
014200      IF DB-STATUS NOT = 0
014300         CALL "DBEXPLAIN" USING DBSTAT
014400         PERFORM CLEANUP
014500         STOP RUN.
```

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.12  COBOL II/3000 FRI, AUG  2, 19

001000$CONTROL USLINIT, LINES = 56, MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. TESTSRCH.
001300 ENVIRONMENT DIVISION.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600      SELECT REPORT-OUTPUT ASSIGN TO "LPFILE,UR".
001700 DATA DIVISION.
001800 FILE SECTION.
001900 FD  REPORT-OUTPUT LABEL RECORDS ARE OMITTED.
002000 01  OUTPUT-AREA PIC X(80).
002100 WORKING-STORAGE SECTION.
002200 01  OUTPUT-REC.
002300     05  PART-NUMBER   PIC X(16).
002400     05  FILLER        PIC X(2) VALUE SPACES.
002500     05  VENDOR-NUMBER PIC Z(4).
002600     05  FILLER        PIC X(2) VALUE SPACES.
002700     05  PURCHASE-DATE PIC X(6).
002800     05  FILLER        PIC X(2) VALUE SPACES.
002900     05  QUANTITY-PURCHASED PIC Z(7)9.99- USAGE DISPLAY.
003000 01  DBSTAT.
003100     05  DB-STATUS PIC S9(4) USAGE COMP.
003200     05  DBSTAT-ELEMENT PIC S9(4) USAGE COMP OCCURS 9.
003300 01 VENDOR-REC.
003400     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
003500     05  VENDOR-NAME   PIC X(20).
003600 01  PURCHASE-REC.
003700     05  PART-NUMBER   PIC X(16).
003800     05  VENDOR-NUMBER PIC S9(4) COMP.
003900     05  PURCHASE-DATE PIC X(6).
004000     05  QUANTITY-PURCHASED PIC S9(8) COMP.
004100 77  DATA-BASE PIC X(16) VALUE "  PURCH  ".
004200 77  VENDOR-MASTER PIC X(16) VALUE "VENDOR-MASTER  ".
004300 77  PURCHASE-FILE PIC X(16) VALUE "PURCHASE-FILE  ".
004400 77  PART-MASTER   PIC X(16) VALUE "PART-MASTER    ".
004500 77  DSET-NAME     PIC X(16).
004600 77  PASSWORD      PIC X(16).
004700 77  IMAGE-LIST    PIC X(32).
004800 77  SAME-LIST     PIC X(2) VALUE "*;".
004900 77  ALL-ITEMS     PIC X(2) VALUE "@;".
005000 01  DUMMY.
005100     05 FILLER    PIC S9(4) USAGE COMP.
005200 77  MODE-1        PIC S9(4) USAGE COMP VALUE 1.
005300 77  MODE-2        PIC S9(4) USAGE COMP VALUE 2.
005400 77  MODE-3        PIC S9(4) USAGE COMP VALUE 3.
005500 77  MODE-4        PIC S9(4) USAGE COMP VALUE 4.
005600 77  MODE-5        PIC S9(4) USAGE COMP VALUE 5.
005700 77  MODE-6        PIC S9(4) USAGE COMP VALUE 6.
005800 77  MODE-7        PIC S9(4) USAGE COMP VALUE 7.
005900 77  DB-MODE       PIC S9(4) USAGE COMP.
006000 77  NAME-INDEX    PIC S9(4) USAGE COMP.
```

```
PAGE 0002/COBTEXT  TESTSRCH
006110*
006120* Search table for vendor number and name
006130*
006140 77 VENDOR-TABLE-SIZE PIC S9(4) USAGE COMP VALUE 0.
006150 01 VENDOR-TABLE.
006160    05 VENDOR-TABLE-ENTRY OCCURS 1 TO 150 TIMES
006170   DEPENDING ON VENDOR-TABLE-SIZE
006180   INDEXED BY VENDOR-INDEX.
006190       10  VENDOR-NUMBER PIC S9(4) USAGE COMP.
006191       10  VENDOR-NAME   PIC X(20).
```

```
PAGE 0003/COBTEXT  TESTSRCH
006200 PROCEDURE DIVISION.
006300 INITIALIZATION SECTION.
006400 INIT-DB.
006500     CALL "PRINTIME".
006600     MOVE "; " TO PASSWORD.
006700     CALL INTRINSIC "DBOPEN" USING DATA-BASE, PASSWORD, MO
006800      DBSTAT.
006900     PERFORM DB-CHECK.
007000     MOVE "VENDOR-MASTER " TO DSET-NAME.
007100     MOVE 201 TO DB-MODE.
007200     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
007300      DBSTAT, VENDOR-MASTER.
007400     PERFORM DB-CHECK.
007500*
007600     MOVE "PURCHASE-FILE " TO DSET-NAME.
007700     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
007800      DBSTAT, PURCHASE-FILE.
007900     PERFORM DB-CHECK.
008000*
008100     MOVE "PART-MASTER " TO DSET-NAME.
008200     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
008300      DBSTAT, PART-MASTER.
008400*
008500* Set up list for PURCHASE-FILE.
008600*
008700     CALL INTRINSIC "DBGET" USING DATA-BASE, PURCHASE-FILE
008800      MODE-1, DBSTAT, ALL-ITEMS, PURCHASE-REC,
008900      DUMMY.
009000     IF DB-STATUS IS NOT = 17 THEN PERFORM DB-CHECK.
009100*
009200* Set up list for VENDOR-MASTER
009300*
009400     MOVE "VENDOR-NUMBER,VENDOR-NAME;" TO IMAGE-LIST.
009500    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
009600    DBSTAT, IMAGE-LIST, PURCHASE-REC, DUMMY.
009700     IF DB-STATUS IS NOT = 17 PERFORM DB-CHECK.
009800*
009900* Open report file
010000*
010100     OPEN OUTPUT REPORT-OUTPUT.
010200      MOVE ALL SPACES to OUTPUT-AREA.
010210*
010220* Initialization for vendor search table.
010230*
010240 INIT-VENDOR-SEARCH.
010250     MOVE 0 TO VENDOR-TABLE-SIZE.
010260 INIT-VENDOR-SEARCH-1.
010270    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
010280           DBSTAT, SAME-LIST, VENDOR-REC, DUMMY
010290     IF DB-STATUS IS = 0
010291         ADD 1 TO VENDOR-TABLE-SIZE
010292         SET VENDOR-INDEX TO VENDOR-TABLE-SIZE
010293         MOVE CORRESPONDING VENDOR-REC TO
```

```
PAGE 0004/COBTEXT  TESTSRCH
00116  010294           VENDOR-TABLE-ENTRY (VENDOR-INDEX)
00117  010295          GO TO INIT-VENDOR-SEARCH-1.
00118  010296      IF DB-STATUS IS NOT = 11 PERFORM DB-CHECK.
00119  010297      CALL "PRINTIME".
00120  010298*
00121  010299* End of initialization.
00122  010300*
```

```
PAGE 0005/COBTEXT  TESTSRCH
010800 REPORT-GENERATION SECTION.
010900 GET-NEXT-RECORD.
011000      CALL "DBGET" USING DATA-BASE, PURCHASE-FILE, MODE-2,
011100      SAME-LIST, PURCHASE-REC, DUMMY.
011200      IF DB-STATUS = 11 GO TO CLEANUP.
011300      PERFORM DB-CHECK.
011400      PERFORM GET-VENDOR.
011500      PERFORM PRINT-LINE.
011600      GO TO GET-NEXT-RECORD.
011700*
011800 PRINT-LINE.
011900      MOVE CORRESPONDING PURCHASE-REC TO OUTPUT-REC.
012000      MOVE CORRESPONDING VENDOR-REC TO OUTPUT-REC.
012100      MOVE OUTPUT-REC TO OUTPUT-AREA.
012200      WRITE OUTPUT-AREA.
012300*
012400 GET-VENDOR.
012500*
012600* Get the vendor specified in the purchase record. To do s
012700* search the vendor table.
012800*
012900     SET VENDOR-INDEX TO 1.
013000     SEARCH VENDOR-TABLE-ENTRY; AT END
013100         DISPLAY "Expected vendor not found - "
013200          VENDOR-NUMBER OF PURCHASE-REC
013210         GO TO CLEANUP
013220      ; WHEN VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX)
013230      VENDOR-NUMBER OF PURCHASE-REC
013240        NEXT SENTENCE.
013250     MOVE VENDOR-NAME OF VENDOR-TABLE (VENDOR-INDEX)
013260     TO VENDOR-NAME OF VENDOR-REC.
013270     MOVE VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX) TO
013280    VENDOR-NUMBER OF VENDOR-REC.
013300*
013400 CLEANUP.
013500      CALL INTRINSIC "DBCLOSE" USING DATA-BASE, DSET-NAME,
013600       MODE-1, DBSTAT.
013700      CLOSE REPORT-OUTPUT.
013800      CALL "PRINTIME".
013900      STOP RUN.
014000*
014100 DB-CHECK.
014200      IF DB-STATUS NOT = 0
014300         CALL "DBEXPLAIN" USING DBSTAT
014400         PERFORM CLEANUP
014500         STOP RUN.
```

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.12  COBOL II/3000 FRI, AUG  2, 19

001000$CONTROL USLINIT, LINES = 56, MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. TESTSRCH.
001300 ENVIRONMENT DIVISION.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600      SELECT REPORT-OUTPUT ASSIGN TO "LPFILE,UR".
001700 DATA DIVISION.
001800 FILE SECTION.
001900 FD  REPORT-OUTPUT LABEL RECORDS ARE OMITTED.
002000 01  OUTPUT-AREA PIC X(80).
002100 WORKING-STORAGE SECTION.
002200 01  OUTPUT-REC.
002300     05  PART-NUMBER   PIC X(16).
002400     05  FILLER        PIC X(2) VALUE SPACES.
002500     05  VENDOR-NUMBER PIC Z(4).
002600     05  FILLER        PIC X(2) VALUE SPACES.
002700     05  PURCHASE-DATE PIC X(6).
002800     05  FILLER        PIC X(2) VALUE SPACES.
002900     05  QUANTITY-PURCHASED PIC Z(7)9.99- USAGE DISPLAY.
003000 01  DBSTAT.
003100     05  DB-STATUS PIC S9(4) USAGE COMP.
003200     05  DBSTAT-ELEMENT PIC S9(4) USAGE COMP OCCURS 9.
003300 01 VENDOR-REC.
003400     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
003500     05  VENDOR-NAME   PIC X(20).
003600 01  PURCHASE-REC.
003700     05  PART-NUMBER   PIC X(16).
003800     05  VENDOR-NUMBER PIC S9(4) COMP.
003900     05  PURCHASE-DATE PIC X(6).
004000     05  QUANTITY-PURCHASED PIC S9(8) COMP.
004100 77  DATA-BASE PIC X(16) VALUE "  PURCH  ".
004200 77  VENDOR-MASTER PIC X(16) VALUE "VENDOR-MASTER   ".
004300 77  PURCHASE-FILE PIC X(16) VALUE "PURCHASE-FILE   ".
004400 77  PART-MASTER   PIC X(16) VALUE "PART-MASTER     ".
004500 77  DSET-NAME     PIC X(16).
004600 77  PASSWORD      PIC X(16).
004700 77  IMAGE-LIST    PIC X(32).
004800 77  SAME-LIST     PIC X(2) VALUE "*;".
004900 77  ALL-ITEMS     PIC X(2) VALUE "@;".
005000 01  DUMMY.
005100     05 FILLER    PIC S9(4) USAGE COMP.
005200 77  MODE-1        PIC S9(4) USAGE COMP VALUE 1.
005300 77  MODE-2        PIC S9(4) USAGE COMP VALUE 2.
005400 77  MODE-3        PIC S9(4) USAGE COMP VALUE 3.
005500 77  MODE-4        PIC S9(4) USAGE COMP VALUE 4.
005600 77  MODE-5        PIC S9(4) USAGE COMP VALUE 5.
005700 77  MODE-6        PIC S9(4) USAGE COMP VALUE 6.
005800 77  MODE-7        PIC S9(4) USAGE COMP VALUE 7.
005900 77  DB-MODE       PIC S9(4) USAGE COMP.
006000 77  NAME-INDEX    PIC S9(4) USAGE COMP.
```

```
PAGE 0002/COBTEXT  TESTSRCH
006200*
006300* Search table for vendor number and name
006400*
006500 77  VENDOR-TABLE-SIZE PIC S9(4) USAGE COMP VALUE 0.
006600 01  VENDOR-TABLE.
006700     05 VENDOR-TABLE-ENTRY OCCURS 1 TO 150 TIMES
006800   DEPENDING ON VENDOR-TABLE-SIZE
006900   ASCENDING KEY IS VENDOR-NUMBER
007000        OF VENDOR-TABLE-ENTRY
007100   INDEXED BY VENDOR-INDEX.
007200        10  VENDOR-NUMBER PIC S9(4) USAGE COMP.
007300        10  VENDOR-NAME   PIC X(20).
007400 77  I PIC S9(4) USAGE COMP.
007500 77  J PIC S9(4) USAGE COMP.
007600 77  K PIC S9(4) USAGE COMP.
007610 77  D PIC S9(4) USAGE COMP.
007700 77  T PIC S9(4) USAGE COMP.
007800 77  S PIC S9(4) USAGE COMP.
007900 01  T-VENDOR-TABLE-ENTRY.
008000     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
008100     05  VENDOR-NAME   PIC X(20).
```

```
PAGE 0003/COBTEXT  TESTSRCH
008300 PROCEDURE DIVISION.
008400 INITIALIZATION SECTION.
008500 INIT-DB.
008600     CALL "PRINTIME".
008700     MOVE "; " TO PASSWORD.
008800     CALL INTRINSIC "DBOPEN" USING DATA-BASE, PASSWORD, MO
008900      DBSTAT.
009000     PERFORM DB-CHECK.
009100     MOVE "VENDOR-MASTER " TO DSET-NAME.
009200     MOVE 201 TO DB-MODE.
009300     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009400      DBSTAT, VENDOR-MASTER.
009500     PERFORM DB-CHECK.
009600*
009700     MOVE "PURCHASE-FILE " TO DSET-NAME.
009800     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009900      DBSTAT, PURCHASE-FILE.
010000     PERFORM DB-CHECK.
010100*
010200     MOVE "PART-MASTER " TO DSET-NAME.
010300     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
010400      DBSTAT, PART-MASTER.
010500*
010600* Set up list for PURCHASE-FILE.
010700*
010800     CALL INTRINSIC "DBGET" USING DATA-BASE, PURCHASE-FILE
010900      MODE-1, DBSTAT, ALL-ITEMS, PURCHASE-REC,
011000      DUMMY.
011100     IF DB-STATUS IS NOT = 17 THEN PERFORM DB-CHECK.
011200*
011300* Set up list for VENDOR-MASTER
011400*
011500     MOVE "VENDOR-NUMBER,VENDOR-NAME;" TO IMAGE-LIST.
011600    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
011700    DBSTAT, IMAGE-LIST, PURCHASE-REC, DUMMY.
011800     IF DB-STATUS IS NOT = 17 PERFORM DB-CHECK.
011900*
012000* Open report file
012100*
012200     OPEN OUTPUT REPORT-OUTPUT.
012300      MOVE ALL SPACES to OUTPUT-AREA.
012400*
012500* Initialization for vendor search table.
012600*
012700 INIT-VENDOR-SEARCH.
012800     MOVE 0 TO VENDOR-TABLE-SIZE.
012900 INIT-VENDOR-SEARCH-1.
013000    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
013100            DBSTAT, SAME-LIST, VENDOR-REC, DUMMY
013200     IF DB-STATUS IS = 0
013300         ADD 1 TO VENDOR-TABLE-SIZE
013400         SET VENDOR-INDEX TO VENDOR-TABLE-SIZE
013500         MOVE CORRESPONDING VENDOR-REC TO
```

```
PAGE 0004/COBTEXT  TESTSRCH
013600              VENDOR-TABLE-ENTRY (VENDOR-INDEX)
013700             GO TO INIT-VENDOR-SEARCH-1.
013800         IF DB-STATUS IS NOT = 11 PERFORM DB-CHECK.
013900*
014000          PERFORM SORT-VENDOR-TABLE THRU SORT-VENDOR-TABLE-END
014100         CALL "PRINTIME".
014200*
014300* End of initialization.
014400*
```

```
PAGE 0005/COBTEXT  TESTSRCH
014600 REPORT-GENERATION SECTION.
014700 GET-NEXT-RECORD.
014800     CALL "DBGET" USING DATA-BASE, PURCHASE-FILE, MODE-2,
014900     SAME-LIST, PURCHASE-REC, DUMMY.
015000     IF DB-STATUS = 11 GO TO CLEANUP.
015100     PERFORM DB-CHECK.
015200     PERFORM GET-VENDOR.
015300     PERFORM PRINT-LINE.
015400     GO TO GET-NEXT-RECORD.
015500*
015600 PRINT-LINE.
015700     MOVE CORRESPONDING PURCHASE-REC TO OUTPUT-REC.
015800     MOVE CORRESPONDING VENDOR-REC TO OUTPUT-REC.
015900     MOVE OUTPUT-REC TO OUTPUT-AREA.
016000     WRITE OUTPUT-AREA.
016100*
016200 GET-VENDOR.
016300*
016400* Get the vendor specified in the purchase record. To do a
016500* search the vendor table.
016600*
016700    SEARCH VENDOR-TABLE-ENTRY; AT END
016800        DISPLAY "Expected vendor not found - "
016900         VENDOR-NUMBER OF PURCHASE-REC
017000        GO TO CLEANUP
017100     ; WHEN VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX)
017200     VENDOR-NUMBER OF PURCHASE-REC
017300       NEXT SENTENCE.
017400    MOVE VENDOR-NAME OF VENDOR-TABLE (VENDOR-INDEX)
017500    TO VENDOR-NAME OF VENDOR-REC.
017600    MOVE VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX) TO
017700   VENDOR-NUMBER OF VENDOR-REC.
017800*
017900 CLEANUP.
018000     CALL INTRINSIC "DBCLOSE" USING DATA-BASE, DSET-NAME,
018100      MODE-1, DBSTAT.
018200     CLOSE REPORT-OUTPUT.
018300     CALL "PRINTIME".
018400     STOP RUN.
018500*
018600 DB-CHECK.
018700     IF DB-STATUS NOT = 0
018800        CALL "DBEXPLAIN" USING DBSTAT
018900        PERFORM CLEANUP
019000        STOP RUN.
```

```
PAGE 0006/COBTEXT  TESTSRCH
019200 SORT-VENDOR-TABLE.
019300*
019400* Sort the vendor lookup table so that a "SEARCH ALL "
019500* statement can be used. (Shell sort.)
019600*
019700     COMPUTE D = 8191.
019800     COMPUTE K = 1.
019900 SORT-K.
020000     IF K > 12 GO TO SORT-K-OUT.
020100     COMPUTE D = (D - 1) /2.
020200     COMPUTE I = D + 1.
020300 SORT-I.
020400     IF I > VENDOR-TABLE-SIZE GO TO SORT-I-OUT.
020500     MOVE VENDOR-TABLE-ENTRY (I) TO T-VENDOR-TABLE-ENTRY.
020600     MOVE VENDOR-NUMBER OF T-VENDOR-TABLE-ENTRY TO T.
020700     COMPUTE J = I - D.
020800 SORT-J.
020900     IF J < 1 GO TO SORT-J-OUT.
021000     IF T NOT < VENDOR-NUMBER OF VENDOR-TABLE-ENTRY ( J )
021100         GO TO SORT-J-OUT.
021200     COMPUTE S = J + D.
021300     MOVE VENDOR-TABLE-ENTRY (J) TO VENDOR-TABLE-ENTRY (S)
021400     COMPUTE J = J - D.
021500     GO TO SORT-J.
021600 SORT-J-OUT.
021700     COMPUTE S = J + D.
021800     MOVE T-VENDOR-TABLE-ENTRY TO VENDOR-TABLE-ENTRY (S).
021900     ADD 1 TO I.
022000     GO TO SORT-I.
022100 SORT-I-OUT.
022200     ADD 1 TO K.
022300     GO TO SORT-K.
022400 SORT-K-OUT.
022500*
022600* Done with sort.
022700*
022800     SET VENDOR-INDEX TO 1.
022900 SORT-PRINT.
023000     IF VENDOR-NUMBER OF VENDOR-TABLE-ENTRY (VENDOR-INDEX)
023100         NOT < VENDOR-NUMBER OF
023200           VENDOR-TABLE-ENTRY (VENDOR-INDEX + 1)
023300        DISPLAY "Sort failed at ", VENDOR-NUMBER OF
023400      VENDOR-TABLE-ENTRY (VENDOR-INDEX)
023500        STOP RUN.
023600     SET VENDOR-INDEX UP BY 1.
023700     IF VENDOR-INDEX < VENDOR-TABLE-SIZE GO TO SORT-PRINT.
023800 SORT-VENDOR-TABLE-END.
```

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.12  COBOL II/3000 FRI, AUG  2, 19

001000$CONTROL USLINIT, LINES = 56, MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. TESTSRCH.
001300 ENVIRONMENT DIVISION.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600      SELECT REPORT-OUTPUT ASSIGN TO "LPFILE,UR".
001700 DATA DIVISION.
001800 FILE SECTION.
001900 FD  REPORT-OUTPUT LABEL RECORDS ARE OMITTED.
002000 01  OUTPUT-AREA PIC X(80).
002100 WORKING-STORAGE SECTION.
002200 01  OUTPUT-REC.
002300         05  PART-NUMBER   PIC X(16).
002400         05  FILLER        PIC X(2) VALUE SPACES.
002500         05  VENDOR-NUMBER PIC Z(4).
002600         05  FILLER        PIC X(2) VALUE SPACES.
002700         05  PURCHASE-DATE PIC X(6).
002800         05  FILLER        PIC X(2) VALUE SPACES.
002900         05  QUANTITY-PURCHASED PIC Z(7)9.99- USAGE DISPLAY.
003000 01  DBSTAT.
003100         05  DB-STATUS PIC S9(4) USAGE COMP.
003200         05  DBSTAT-ELEMENT PIC S9(4) USAGE COMP OCCURS 9.
003300 01 VENDOR-REC.
003400         05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
003500         05  VENDOR-NAME   PIC X(20).
003600 01  PURCHASE-REC.
003700         05  PART-NUMBER   PIC X(16).
003800         05  VENDOR-NUMBER PIC S9(4) COMP.
003900         05  PURCHASE-DATE PIC X(6).
004000         05  QUANTITY-PURCHASED PIC S9(8) COMP.
004100 77  DATA-BASE PIC X(16) VALUE "  PURCH  ".
004200 77  VENDOR-MASTER PIC X(16) VALUE "VENDOR-MASTER   ".
004300 77  PURCHASE-FILE PIC X(16) VALUE "PURCHASE-FILE   ".
004400 77  PART-MASTER   PIC X(16) VALUE "PART-MASTER     ".
004500 77  DSET-NAME     PIC X(16).
004600 77  PASSWORD      PIC X(16).
004700 77  IMAGE-LIST    PIC X(32).
004800 77  SAME-LIST     PIC X(2) VALUE "*;".
004900 77  ALL-ITEMS     PIC X(2) VALUE "@;".
005000 01  DUMMY.
005100     05 FILLER    PIC S9(4) USAGE COMP.
005200 77  MODE-1        PIC S9(4) USAGE COMP VALUE 1.
005300 77  MODE-2        PIC S9(4) USAGE COMP VALUE 2.
005400 77  MODE-3        PIC S9(4) USAGE COMP VALUE 3.
005500 77  MODE-4        PIC S9(4) USAGE COMP VALUE 4.
005600 77  MODE-5        PIC S9(4) USAGE COMP VALUE 5.
005700 77  MODE-6        PIC S9(4) USAGE COMP VALUE 6.
005800 77  MODE-7        PIC S9(4) USAGE COMP VALUE 7.
005900 77  DB-MODE       PIC S9(4) USAGE COMP.
006000 77  NAME-INDEX    PIC S9(4) USAGE COMP.
```

```
PAGE 0002/COBTEXT  TESTSRCH
006200*
006300* Search table for vendor number and name
006400*
006500 77  VENDOR-TABLE-SIZE PIC S9(4) USAGE COMP VALUE 0.
006600 01  VENDOR-TABLE.
006700     05 VENDOR-TABLE-ENTRY OCCURS 1 TO 150 TIMES
006800   DEPENDING ON VENDOR-TABLE-SIZE
006900   ASCENDING KEY IS VENDOR-NUMBER
007000        OF VENDOR-TABLE-ENTRY
007100   INDEXED BY VENDOR-INDEX.
007200        10  VENDOR-NUMBER PIC S9(4) USAGE COMP.
007300        10  VENDOR-NAME   PIC X(20).
007400 77  I PIC S9(4) USAGE COMP.
007500 77  J PIC S9(4) USAGE COMP.
007600 77  K PIC S9(4) USAGE COMP.
007610 77  D PIC S9(4) USAGE COMP.
007700 77  T PIC S9(4) USAGE COMP.
007800 77  S PIC S9(4) USAGE COMP.
007900 01  T-VENDOR-TABLE-ENTRY.
008000     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
008100     05  VENDOR-NAME   PIC X(20).
```

```
PAGE 0003/COBTEXT  TESTSRCH
008300 PROCEDURE DIVISION.
008400 INITIALIZATION SECTION.
008500 INIT-DB.
008600     CALL "PRINTIME".
008700     MOVE "; " TO PASSWORD.
008800     CALL INTRINSIC "DBOPEN" USING DATA-BASE, PASSWORD, MO
008900      DBSTAT.
009000     PERFORM DB-CHECK.
009100     MOVE "VENDOR-MASTER " TO DSET-NAME.
009200     MOVE 201 TO DB-MODE.
009300     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009400      DBSTAT, VENDOR-MASTER.
009500     PERFORM DB-CHECK.
009600*
009700     MOVE "PURCHASE-FILE " TO DSET-NAME.
009800     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009900      DBSTAT, PURCHASE-FILE.
010000     PERFORM DB-CHECK.
010100*
010200     MOVE "PART-MASTER " TO DSET-NAME.
010300     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
010400      DBSTAT, PART-MASTER.
010500*
010600* Set up list for PURCHASE-FILE.
010700*
010800     CALL INTRINSIC "DBGET" USING DATA-BASE, PURCHASE-FILE
010900      MODE-1, DBSTAT, ALL-ITEMS, PURCHASE-REC,
011000      DUMMY.
011100     IF DB-STATUS IS NOT = 17 THEN PERFORM DB-CHECK.
011200*
011300* Set up list for VENDOR-MASTER
011400*
011500     MOVE "VENDOR-NUMBER,VENDOR-NAME;" TO IMAGE-LIST.
011600    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
011700    DBSTAT, IMAGE-LIST, PURCHASE-REC, DUMMY.
011800     IF DB-STATUS IS NOT = 17 PERFORM DB-CHECK.
011900*
012000* Open report file
012100*
012200     OPEN OUTPUT REPORT-OUTPUT.
012300      MOVE ALL SPACES to OUTPUT-AREA.
012400*
012500* Initialization for vendor search table.
012600*
012700 INIT-VENDOR-SEARCH.
012800     MOVE 0 TO VENDOR-TABLE-SIZE.
012900 INIT-VENDOR-SEARCH-1.
013000    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
013100            DBSTAT, SAME-LIST, VENDOR-REC, DUMMY
013200     IF DB-STATUS IS = 0
013300         ADD 1 TO VENDOR-TABLE-SIZE
013400         SET VENDOR-INDEX TO VENDOR-TABLE-SIZE
013500         MOVE CORRESPONDING VENDOR-REC TO
```

```
PAGE 0004/COBTEXT  TESTSRCH
013600            VENDOR-TABLE-ENTRY (VENDOR-INDEX)
013700           GO TO INIT-VENDOR-SEARCH-1.
013800       IF DB-STATUS IS NOT = 11 PERFORM DB-CHECK.
013900*
014000        PERFORM SORT-VENDOR-TABLE THRU SORT-VENDOR-TABLE-END
014100       CALL "PRINTIME".
014200*
014300* End of initialization.
014400*
```

```
PAGE 0005/COBTEXT  TESTSRCH
014600 REPORT-GENERATION SECTION.
014700 GET-NEXT-RECORD.
014800      CALL "DBGET" USING DATA-BASE, PURCHASE-FILE, MODE-2,
014900      SAME-LIST, PURCHASE-REC, DUMMY.
015000      IF DB-STATUS = 11 GO TO CLEANUP.
015100      PERFORM DB-CHECK.
015200      PERFORM GET-VENDOR.
015300      PERFORM PRINT-LINE.
015400      GO TO GET-NEXT-RECORD.
015500*
015600 PRINT-LINE.
015700      MOVE CORRESPONDING PURCHASE-REC TO OUTPUT-REC.
015800      MOVE CORRESPONDING VENDOR-REC TO OUTPUT-REC.
015900      MOVE OUTPUT-REC TO OUTPUT-AREA.
016000      WRITE OUTPUT-AREA.
016100*
016200 GET-VENDOR.
016300*
016400* Get the vendor specified in the purchase record. To do s
016500* search the vendor table.
016600*
016700     SEARCH VENDOR-TABLE-ENTRY; AT END
016800         DISPLAY "Expected vendor not found - "
016900          VENDOR-NUMBER OF PURCHASE-REC
017000         GO TO CLEANUP
017100      ; WHEN VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX)
017200      VENDOR-NUMBER OF PURCHASE-REC
017300        NEXT SENTENCE.
017400     MOVE VENDOR-NAME OF VENDOR-TABLE (VENDOR-INDEX)
017500     TO VENDOR-NAME OF VENDOR-REC.
017600     MOVE VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX) TO
017700    VENDOR-NUMBER OF VENDOR-REC.
017800*
017900 CLEANUP.
018000      CALL INTRINSIC "DBCLOSE" USING DATA-BASE, DSET-NAME,
018100       MODE-1, DBSTAT.
018200      CLOSE REPORT-OUTPUT.
018300      CALL "PRINTIME".
018400      STOP RUN.
018500*
018600 DB-CHECK.
018700      IF DB-STATUS NOT = 0
018800         CALL "DBEXPLAIN" USING DBSTAT
018900         PERFORM CLEANUP
019000         STOP RUN.
```

```
PAGE 0006/COBTEXT  TESTSRCH
019200 SORT-VENDOR-TABLE.
019300*
019400* Sort the vendor lookup table so that a "SEARCH ALL "
019500* statement can be used. (Shell sort.)
019600*
019700     COMPUTE D = 8191.
019800     COMPUTE K = 1.
019900 SORT-K.
020000     IF K > 12 GO TO SORT-K-OUT.
020100     COMPUTE D = (D - 1) /2.
020200     COMPUTE I = D + 1.
020300 SORT-I.
020400     IF I > VENDOR-TABLE-SIZE GO TO SORT-I-OUT.
020500     MOVE VENDOR-TABLE-ENTRY (I) TO T-VENDOR-TABLE-ENTRY.
020600     MOVE VENDOR-NUMBER OF T-VENDOR-TABLE-ENTRY TO T.
020700     COMPUTE J = I - D.
020800 SORT-J.
020900     IF J < 1 GO TO SORT-J-OUT.
021000     IF T NOT < VENDOR-NUMBER OF VENDOR-TABLE-ENTRY ( J )
021100         GO TO SORT-J-OUT.
021200     COMPUTE S = J + D.
021300     MOVE VENDOR-TABLE-ENTRY (J) TO VENDOR-TABLE-ENTRY (S)
021400     COMPUTE J = J - D.
021500     GO TO SORT-J.
021600 SORT-J-OUT.
021700     COMPUTE S = J + D.
021800     MOVE T-VENDOR-TABLE-ENTRY TO VENDOR-TABLE-ENTRY (S).
021900     ADD 1 TO I.
022000     GO TO SORT-I.
022100 SORT-I-OUT.
022200     ADD 1 TO K.
022300     GO TO SORT-K.
022400 SORT-K-OUT.
022500*
022600* Done with sort.
022700*
022800     SET VENDOR-INDEX TO 1.
022900 SORT-PRINT.
023000     IF VENDOR-NUMBER OF VENDOR-TABLE-ENTRY (VENDOR-INDEX)
023100         NOT < VENDOR-NUMBER OF
023200           VENDOR-TABLE-ENTRY (VENDOR-INDEX + 1)
023300        DISPLAY "Sort failed at ", VENDOR-NUMBER OF
023400      VENDOR-TABLE-ENTRY (VENDOR-INDEX)
023500        STOP RUN.
023600     SET VENDOR-INDEX UP BY 1.
023700     IF VENDOR-INDEX < VENDOR-TABLE-SIZE GO TO SORT-PRINT.
023800 SORT-VENDOR-TABLE-END.
```

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.12  COBOL II/3000 FRI, AUG  2, 19

001000$CONTROL USLINIT, LINES = 56, MAP
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. TESTSRCH.
001300 ENVIRONMENT DIVISION.
001400 INPUT-OUTPUT SECTION.
001500 FILE-CONTROL.
001600      SELECT REPORT-OUTPUT ASSIGN TO "LPFILE,UR".
001700 DATA DIVISION.
001800 FILE SECTION.
001900 FD  REPORT-OUTPUT LABEL RECORDS ARE OMITTED.
002000 01  OUTPUT-AREA PIC X(80).
002100 WORKING-STORAGE SECTION.
002200 01  OUTPUT-REC.
002300     05  PART-NUMBER    PIC X(16).
002400     05  FILLER         PIC X(2) VALUE SPACES.
002500     05  VENDOR-NUMBER PIC Z(4).
002600     05  FILLER         PIC X(2) VALUE SPACES.
002700     05  PURCHASE-DATE PIC X(6).
002800     05  FILLER         PIC X(2) VALUE SPACES.
002900     05  QUANTITY-PURCHASED PIC Z(7)9.99- USAGE DISPLAY.
003000 01  DBSTAT.
003100     05  DB-STATUS PIC S9(4) USAGE COMP.
003200     05  DBSTAT-ELEMENT PIC S9(4) USAGE COMP OCCURS 9.
003300 01 VENDOR-REC.
003400     05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
003500     05  VENDOR-NAME    PIC X(20).
003600 01  PURCHASE-REC.
003700     05  PART-NUMBER    PIC X(16).
003800     05  VENDOR-NUMBER PIC S9(4) COMP.
003900     05  PURCHASE-DATE PIC X(6).
004000     05  QUANTITY-PURCHASED PIC S9(8) COMP.
004100 77  DATA-BASE PIC X(16) VALUE "  PURCH  ".
004200 77  VENDOR-MASTER PIC X(16) VALUE "VENDOR-MASTER   ".
004300 77  PURCHASE-FILE PIC X(16) VALUE "PURCHASE-FILE   ".
004400 77  PART-MASTER    PIC X(16) VALUE "PART-MASTER     ".
004500 77  DSET-NAME      PIC X(16).
004600 77  PASSWORD       PIC X(16).
004700 77  IMAGE-LIST     PIC X(32).
004800 77  SAME-LIST      PIC X(2) VALUE "*;".
004900 77  ALL-ITEMS      PIC X(2) VALUE "@;".
005000 01  DUMMY.
005100     05 FILLER     PIC S9(4) USAGE COMP.
005200 77  MODE-1         PIC S9(4) USAGE COMP VALUE 1.
005300 77  MODE-2         PIC S9(4) USAGE COMP VALUE 2.
005400 77  MODE-3         PIC S9(4) USAGE COMP VALUE 3.
005500 77  MODE-4         PIC S9(4) USAGE COMP VALUE 4.
005600 77  MODE-5         PIC S9(4) USAGE COMP VALUE 5.
005700 77  MODE-6         PIC S9(4) USAGE COMP VALUE 6.
005800 77  MODE-7         PIC S9(4) USAGE COMP VALUE 7.
005900 77  DB-MODE        PIC S9(4) USAGE COMP.
006000 77  NAME-INDEX     PIC S9(4) USAGE COMP.
```

```
PAGE 0002/COBTEXT  TESTSRCH
006200*
006300* Search table for vendor number and name
006400*
006500 77  VENDOR-TABLE-SIZE PIC S9(4) USAGE COMP VALUE 0.
006600 01  VENDOR-TABLE.
006700         05 VENDOR-TABLE-ENTRY OCCURS 1 TO 150 TIMES
006800   DEPENDING ON VENDOR-TABLE-SIZE
006900   INDEXED BY VENDOR-INDEX.
007000            10  VENDOR-NUMBER PIC S9(4) USAGE COMP.
007100            10  VENDOR-NAME   PIC X(20).
007200 01  TEMP-VENDOR-TABLE-ENTRY.
007300         05  VENDOR-NUMBER PIC S9(4) USAGE COMP.
007400         05  VENDOR-NAME   PIC X(20).
```

```
PAGE 0003/COBTEXT  TESTSRCH
007600 PROCEDURE DIVISION.
007700 INITIALIZATION SECTION.
007800 INIT-DB.
007900     CALL "PRINTIME".
008000     MOVE "; " TO PASSWORD.
008100     CALL INTRINSIC "DBOPEN" USING DATA-BASE, PASSWORD, MO
008200      DBSTAT.
008300     PERFORM DB-CHECK.
008400     MOVE "VENDOR-MASTER " TO DSET-NAME.
008500     MOVE 201 TO DB-MODE.
008600     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
008700      DBSTAT, VENDOR-MASTER.
008800     PERFORM DB-CHECK.
008900*
009000     MOVE "PURCHASE-FILE " TO DSET-NAME.
009100     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009200      DBSTAT, PURCHASE-FILE.
009300     PERFORM DB-CHECK.
009400*
009500     MOVE "PART-MASTER " TO DSET-NAME.
009600     CALL INTRINSIC "DBINFO" USING DATA-BASE, DSET-NAME, D
009700      DBSTAT, PART-MASTER.
009800*
009900* Set up list for PURCHASE-FILE.
010000*
010100     CALL INTRINSIC "DBGET" USING DATA-BASE, PURCHASE-FILE
010200      MODE-1, DBSTAT, ALL-ITEMS, PURCHASE-REC,
010300      DUMMY.
010400     IF DB-STATUS IS NOT = 17 THEN PERFORM DB-CHECK.
010500*
010600* Set up list for VENDOR-MASTER
010700*
010800     MOVE "VENDOR-NUMBER,VENDOR-NAME;" TO IMAGE-LIST.
010900    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
011000    DBSTAT, IMAGE-LIST, PURCHASE-REC, DUMMY.
011100     IF DB-STATUS IS NOT = 17 PERFORM DB-CHECK.
011200*
011300* Open report file
011400*
011500     OPEN OUTPUT REPORT-OUTPUT.
011600      MOVE ALL SPACES to OUTPUT-AREA.
011700*
011800* Initialization for vendor search table.
011900*
012000 INIT-VENDOR-SEARCH.
012100     MOVE 0 TO VENDOR-TABLE-SIZE.
012200 INIT-VENDOR-SEARCH-1.
012300    CALL INTRINSIC "DBGET" USING DATA-BASE, VENDOR-MASTER,
012400            DBSTAT, SAME-LIST, VENDOR-REC, DUMMY
012500     IF DB-STATUS IS = 0
012600         ADD 1 TO VENDOR-TABLE-SIZE
012700         SET VENDOR-INDEX TO VENDOR-TABLE-SIZE
012800         MOVE CORRESPONDING VENDOR-REC TO
```

```
PAGE 0004/COBTEXT  TESTSRCH
012900           VENDOR-TABLE-ENTRY (VENDOR-INDEX)
013000          GO TO INIT-VENDOR-SEARCH-1.
013100      IF DB-STATUS IS NOT = 11 PERFORM DB-CHECK.
013200      CALL "PRINTIME".
013300*
013400* End of initialization.
013500*
```

```
PAGE 0005/COBTEXT  TESTSRCH
013520 REPORT-GENERATION SECTION.
013530 GET-NEXT-RECORD.
013540      CALL "DBGET" USING DATA-BASE, PURCHASE-FILE, MODE-2,
013550      SAME-LIST, PURCHASE-REC, DUMMY.
013560      IF DB-STATUS = 11 GO TO CLEANUP.
013570      PERFORM DB-CHECK.
013580      PERFORM GET-VENDOR.
013590      PERFORM PRINT-LINE.
013600      GO TO GET-NEXT-RECORD.
013610*
013620 PRINT-LINE.
013630      MOVE CORRESPONDING PURCHASE-REC TO OUTPUT-REC.
013640      MOVE CORRESPONDING VENDOR-REC TO OUTPUT-REC.
013650      MOVE OUTPUT-REC TO OUTPUT-AREA.
013660      WRITE OUTPUT-AREA.
013670*
013680 GET-VENDOR.
013690*
013700* Get the vendor specified in the purchase record. To do s
013710* search the vendor table.
013720*
013730     SET VENDOR-INDEX TO 1.
013740     SEARCH VENDOR-TABLE-ENTRY; AT END
013750         DISPLAY "Expected vendor not found - "
013760          VENDOR-NUMBER OF PURCHASE-REC
013770         GO TO CLEANUP
013780      ; WHEN VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX)
013790      VENDOR-NUMBER OF PURCHASE-REC
013800        NEXT SENTENCE.
013810     MOVE VENDOR-NAME OF VENDOR-TABLE (VENDOR-INDEX)
013820     TO VENDOR-NAME OF VENDOR-REC.
013830     MOVE VENDOR-NUMBER OF VENDOR-TABLE (VENDOR-INDEX) TO
013840    VENDOR-NUMBER OF VENDOR-REC.
013850*
013860* Also, interchange the located entry with the preceding o
013870* unless it's already the first or second.
013880*
013890     IF VENDOR-INDEX > 2
013900        MOVE VENDOR-TABLE-ENTRY (VENDOR-INDEX) TO
013910  TEMP-VENDOR-TABLE-ENTRY
013920        MOVE VENDOR-TABLE-ENTRY (VENDOR-INDEX - 1)
013930  TO VENDOR-TABLE-ENTRY (VENDOR-INDEX)
013940        MOVE TEMP-VENDOR-TABLE-ENTRY TO
013950  VENDOR-TABLE-ENTRY (VENDOR-INDEX - 1).
013960*
013970 CLEANUP.
013980      CALL INTRINSIC "DBCLOSE" USING DATA-BASE, DSET-NAME,
013990       MODE-1, DBSTAT.
014000      CLOSE REPORT-OUTPUT.
014010      CALL "PRINTIME".
014020      STOP RUN.
014030*
014040 DB-CHECK.
```

```
PAGE 0006/COBTEXT  TESTSRCH
014050        IF DB-STATUS NOT = 0
014060           CALL "DBEXPLAIN" USING DBSTAT
014070           PERFORM CLEANUP
014080           STOP RUN.
```

3054. HEWLETT PACKARD RESPONSE CENTER

Roy A. Clifton
Support Operations Manager
HP North American Response Center
Santa Clara, California

I. Original Concept

- A. The problem statement
- B. Locations
- C. Staffing
- D. DP support
- E. Productivity improvements
- F. Response Center Objectives

II. The first year

- A. Staffing
- B. HPCOACH
- C. DP support
- D. Remote Support
- E. RC Documentation
- F. Patch Coordination

III. The second year and beyond

- A. Predictive Support
- B. After Hours Dispatch
- C. CE Technical Assist
- D. Total Quality Control
- E. System Support Team
- F. System Interrupt Team
- G. Application Notes
- H. Product Division Feedback
- I. Future RC Tools

IV. Summary

V. Statistical information

This is the written presentation text; the actual presentation will focus less upon the Response Center evolution and more upon the statistical results of its existence.

I. Original Concept

The current implementation of Hewlett Packard's Response Centers is, in some ways, different than described below. In most ways it is the same. Notably, the European Response Center has evolved into a two-layer organization, the Country Response Centers and Computer Support Pinewood rather than as a single entity. The Original Concept is presented to show how the program was initially envisioned.

A. The problem statement

Prior to the Response Centers (RC) HP supplied telephone support (PICS) to customers on a contractual basis at the local (area/country) level. This part of our software support program had been extremely important in achieving both customer satisfaction and customer loyalty. At the same time it became a limiting factor in our ability to expand our support offerings. The number of Systems Engineers dedicated to telephone support continued to increase but not in proportion to the number of products that had to be supported. To prepare for the future support requirements of our customers we needed to consolidate our current PICS activities into two Response Centers in North America and one Response Center in Europe.

These Response Centers would encompass our current AEO phone-in-consulting as well as the hardware Tele-support and hardware after-hours dispatch. Each center would consist of a day shift and two off-hours shifts thus allowing 24 hour operation. The average permanent staffing of each center would be 60 professionals, 12 technical support, and nine managers. In addition to the permanent staff, approximately 10 field profesisonals would be involved on a rotational basis with the Response Center.

Two centers in North America would provide support to ICON PICs Centers. The European Center would be self-contained with its own data base, while the two centers in North America would share a single data base. Information common to both systems would be kept in sync via daily DS transmissions.

The implementation of the Response Centers would start in October of FY '83 and be completed by February of FY '84. The two centers in North America would begin customer support in Q2 FY '84. The European Center would lag North America by approximately one quarter due to the more complex issues involved in data communication. The described implementation was chosen to minimize the impact of their transition on the field operations and the financial start-up costs.

B. Locations

The location of the Response Centers has been evaluated and the following sites recommended -

1. Atlanta - (ARC) given the percentage of HP customers in the Eastern Time Zone one of the two North American Centers should be in that zone. Atlanta was chosen because of it being easy to attract people to live there; as well as having good schools and direct air transportation to the rest of the U.S. Since Atlanta is also the Southern Region Headquarters it would be easier to leverage some needed services in that location.

2. Cupertino/Sunnyvale/Santa Clara - (SCRC) the second North America site should be close to our Systems divisions and IND and CSD as well as serving the high percentage of customers in the Pacific Time Zone.

3. Europe - Pinewood (ERC) - primary consideration is the availability of skilled professionals and existing support from the current HP site. The cost and availability of communications media, and the UK's membership in the EEC were the deciding factors in choosing Pinewood.

C. Staffing

Staffing and hiring for the Response Centers would take place in a phased method during FY '84. During the first quarter the North American Centers would hire and train approximately 30% of the required professionals. This hiring would be off-set by not hiring the same number of people in the field operations. During Q2 FY '84 the North American Response Centers would begin operation by drafting almost 100% of the needed professionals from the field for two week tours of duty in the centers. Additional staff would be hired by the centers during Q2 FY'84 again offsetting these hires by not hiring field operations professionals. In Q3 the centers would complete their hiring of full time staff. The field contribution would decrease to 70% due to first quarter hires becoming productive. Hiring would return to normal in the field operations during fourth quarter and the field draft would decrease to 30%. By the start of Q1 FY '85 steady state operation is achieved in North America. The centers would be fully staffed and only 10% of the members would be field professionals on two week assignments.

The European Response Center would follow a similar plan but one quarter offset from the North American Centers. Although this offset is due to circumstances outside our control it has the benefit of distributing the start-up costs over the year and allows us to learn from our experiences in Atlanta and Cupertino.

D. DP Support

The Response Centers will be supported for EDP purposes by two data centers - one in Cupertino and one in Europe. Each data center will consist of a large HP3000 system, UPS system, data comm control and spares to allow continuous operation. Initial software will consist of STARSII, IBS, TRAKII and possibly FIREMAN. Atlanta will be connected to Cupertino via leased lines and multi-point. Eventually X.25 connections will be used to lower the communication costs.

E. Productivity Gains

In addition to providing customers with a better level of overall support and service it was expected that we could accomplish the following real productivity gains in the affected organizations - AEO and CEO.

1) Provide 24 hour, 7 day per week software support with between the same to no more than 10% increase in the number of professionals.

2) Provide 24 hour hardware Tele-Support with the same staffing level as current 8-5 coverage.

3) Provide an HP after-hours answering service for hardware maintenance with better quality and consistency than is currently delivered for approximately the same cost as the multiple methods of operation today.

4) Minimize the interrupt driven activities in our field operations.

5) Decrease the duplication of current field operations activity in providing software support.

6) Speed-up the ability to provide quality support on new products.

7) Decrease the amount of field support equipment.

8) Reduce the duplication of effort that takes place in solving similar problems in our field technical centers.

9) Improve our ability to support major accounts with dispersed networks.

F. RC Objectives

The RC objectives established were:

Use technology to increase customer satisfaction through the use of a common database and increased availability of diagnostic tools and predictive maintenance.

Improve teamwork to decrease problem resolution times by creating support teams which cross discipline lines and making them responsible for solving problems.

Leverage off response center to improve support by increasing hours of coverage, identifying frequency of problems, and working with divisions on new support tools and supportability of new products.

II. The First Year

A. Staffing

During the first year of operation, technical staffing for the RCs was accomplished by using HP field engineers on a rotating loan program (FOL). This provided several benefits:

Customers received the best combination of experience and technical knowledge to answer their calls.

RC engineers were provided the opportunity to be trained in a less stressful environment.

RC engineers had the chance to watch and learn from field engineers to broaden their view of interfacing with customers.

Field engineers were exposed to a wider variety of calls than they would normally encounter in their own offices.

As RC engineers (RCEs) completed their training (see "The RCE training binder", below) they were mentored by FOL engineers until ready to proceed upon their own. Note that the training included analytical and people skills training in addition to technical training. At the time RCEs went online, they were typically better trained (technically) than their FOL counterparts, but did not have the equivalent "real" experience. However, this was at least partially offset by the fact that the RCEs were hired specifically for this job and they did not consider this to be a 'part time' job, but rather as their 'real' job. During the latter part of the first year, the FOL contingent in the RC began to ramp down as the permanent engineers began to take on the call responsibility. For the most part, hardware RCEs were already trained CEs from HP's field operations.

In addition to the technical, analytical and people skills training, engineers were expected to participate in programming projects and field and factory visits. As we listened to our FOL engineers we determined that burnout would be a significant issue if not addressed at an early stage. Therefore, we made several important decisions. The first was that we would go light on managment hiring to allow hiring additional engineers early in the program. This would maximize the availability of engineers for phone duty as early as possible. Secondly, we decided to hire more engineers than actually needed to answer the expected volume of calls. This would allow all RCEs to have weeks off the phones. During this time, they would work on projects, visit the field and/or factory, take training or other activities except taking calls. This strategy worked well in providing for a motivated staff of engineers and an extremely low turnover rate. However, the managment team began to experience its own burnout. This was addressed in November, 1984 through a reorganization which provided for an additional 8 managers in each RC. The bulk of the new management team was in place by April, 1985. This provided the right balance of engineers and management for efficient operation.

From the beginning RCEs were organized into mixed discipline groups. This means that each working team of engineers was composed of hardware, general system and applications software members. There were several reasons for this orientation:

> People naturally communicate with others having the same expertise and experiences. This is a comfortable mode. By creating mixed teams, we expected significantly greater exchange between members not having the same background. This would tend to raise the average system-wide knowledge of every engineer rather than to increase their specific expertise.
>
> All the expertise necessary to resolve almost any problem now would reside on each team rather than on any specific team. This would provide for 'trapping' calls in a team and having that team responsible for resolving it. In addition, if expertise in another area was needed, an appropriate engineer would be on the team instead of on another team located at the other end of the building or elsewhere.

Prior to the previously mentioned reorganization, the different 'types' of engineers were separately managed even though they sat together on a team. This means that all hardware engineers were managed by one manager, all applications software engineers were managed by a different manager and the general software engineers were managed by a third and fourth manager. Holding team meetings and coaching and planning sessions was difficult due to the spread-out nature of the members of a particular manager's organization. As an integral feature of the reorganization, the mixed teams are now managed as a team. This means that our first level managers have responsibility for all three 'types' of engineers. The advantage, of course, is that they all sit together as a team.

B. HPCOACH

The PC support organization, HPCOACH, was merged with the RCs in late 1984. This would coalesce HP's support for computer products into one organization. PC product support in Europe is provided through the Country Response Centers.

HPCOACH, originally known as the Personal Software Assistance Center, went "on the air" on October 1, 1983. The initial charter was to assist customers in using PC software from PSD. This charter was soon expanded to include peripheral and systems support for HP's PCs.

In its first month HPCOACH answered 2,735 customer questions. This number grew rapidly until February 1984 became our first 10,000 answer month.

HPCOACH has made a major contribution to the quality of HP's personal computers, peripherals and software by providing fast feedback from users to the various divisions. This was particularly true in the case of PSD where, as part of the PSD Support Group, HPCOACH actually participated in the MR process.

HPCOACH has been a focal point for the production and distribution of documentation for our customers. In January, we produced our first edition of "Answers to the Most Frequently Asked Questions on the HP 150". This document was soon packed with each 150 shipped. It has now been expanded to cover the HP110 and peripherals. HPCOACH has generated over 30 application notes and along with notes created by the product divisions distributes about 50 application notes per day to our customers.

C. DP Support

As identified in section I.D., the RCs opened with one DP facility in Santa Clara with Atlanta achieving access via leased lines. Within a relatively short time, this DP arrangement was determined to be inadequate for the RCs needs. Therefore, additional systems and equipment along with the staffing to operate, configure and program it was added to the function. From a start with one system manager/system administrator the DP organization grew to two development engineers, one programmer/analyst, one diagnostic system manager, one operations manager and one facilities information systems manager along with the necessary operations staff to operate 7 days a week, 24 hours a day within the first year. By the end of that first year, three administrative systems were necessary in addition to the engineer's hands on diagnostic systems to run the business.

D. Remote support (Telesupport)

Remote support was the first hardware support program implemented in the RCs. This function, as well as hardware consulting for the software engineers, is performed by the hardware engineers on each team. The concept is to use remote diagnostic procedures on HP3000 systems to determine failure causes and provide that information to the responsible Customer Engineer (CE) for his use in resolving the problem. The local HP area office determines if a hardware service call can be remotely diagnosed. If so, the call is electronically transferred to the RC for analysis. After the RCE investigates the problem, the results are electronically transferred back to the local office for use by the CE. This procedure saves time for the CE and customer by determining the cause of failures prior to the CE arriving on the customer's site ensuring that the CE has the appropriate parts and expertise. In addition, we have found that about 20% of the calls received in the RC can be resolved without the CE going on site.

E. RC documentation

Our first year of operation was focused upon working out the details of our daily jobs. This was primarily in the areas of interfacing with customers and HP field and factory organizations as well as ourselves. To provide consistency, the following documents were developed:

1. The Book of Knowledge is a compendium of how-tos and check lists for RCEs. Included are items such as factory support organization charts, how to download patches, memory dump procedures and field support office locations and names.

2. The RCE training binder contains a copy of the RC training plan and check lists for completion of course modules. The plan includes technical, analytical and people skills development materials and managers check off courses as they are completed. This document defines a "phones ready" RCE. This document, of course, is in constant refinement as new products and training becomes available and necessary.

3. A Field Notification Procedure defines the process for reassigning responsibility for an RC call to the HP field operation. This must be done in a controlled manner and the process provides a detailed flowchart format to cover all contingencies. This document was developed jointly with the HP field operation.

F. Patch Coordination.

During first year of operation, we found that everyone seemed to benefit by the RCs ability to install patches on customers systems to resolve problems. However, this is a time consuming procedure and, therefore we implemented the patch coordinator. This person(s) was responsible for installing patches on customers systems when requested by an RCE. This program provided consistency in the patch procedure and increased RCE productivity.

III. The second year and beyond

During the first part of the second year of operations, the RCs began to focus more energy on improving existing programs and developing new ways to leverage our organization and knowledge to provide better customer support and lay the groundwork for better future products. In addition, we began implementing product information feedback to the HP product divisions for their use in prioritizing the assignment of their resources.

A. Predictive support

This program provides for programmatic statistical analysis of the customer's log files. Through this analysis, future device failures can be predicted. The customer's system then transmits this information to an HP system at the RC. This automatically envokes Remote Support and possible on-site activity. The major benefit is that repairs can be scheduled prior to failure rather than at the time of failure. The customer schedules the running of the analysis program on their system and thus maintains control of their system utilization. It is anticipated that Predictive Support will interface to future Artificial Intelligence support tools.

B. After Hours Dispatch

Centralized After Hours Dispatching contributes to Hewlett Packard's goal of improving customer satisfaction by providing dispatching service during night business hours in a manner similar to that used during day business hours. Customer support contracts are checked against a database maintained by the customer's local service office and in this way we are able to explain any additional cost that might be associated with on-site service that night. In addition, Remote Support services are extended to assist customers in recovering quickly from system or peripheral malfunctions. If on-site service is required to resolve the problem, the RC will page the appropriate Customer Engineer and he/she will call the customer back to arrange for the visit.

C. CE Technical Assist

Customer Engineers being very responsible individuals, usually know when they need help in understanding and resolving a problem. The Customer Engineer usually just wants to discuss the events, symptoms and his action plan with a more experienced engineer to determine if he is headed in the right direction. The Response Center has started to develop an on-line support organization that we call CE Tech Assist. Currently, the Center has a full staff of engineers who deal with the workstation products. The next product family will be the commercial systems and then the technical systems. This consulting team also collects data relative to the reliability and supportabilty of our products and participates with the manufacturing divisions in improving current products and designing future products.

D. Total Quality Control

Total Quality Control (TQC) is HPs reference to its implementation stategy for a quality program that has the objective to improve customer satisfaction by eliminating defects from any process, whether it be a manufacturing activity or a service. TQC is a managment philosophy totally committed to quality that focuses on continually improving the engineer's work by employing scientific methods of data collection and analysis on the process steps with a goal of perfecting them - perfection begin the elimination of defects or "doing the job right the first time". The result of the TQC methodology is measured by the end product exceeding the customer's expectations.

The US Response Centers are approaching TQC from two different directions. One stresses first line employee involvement in isolating and resolving deviations and bottlenecks in the process of providing answers to customer calls. This approach encourages employee participation and has the benefit of increasing employee morale by giving them responsibility for identifying problems that impede the delivery of the correct solution to Response Center users, proposing solutions to the management of the center and implementing them with management involvement and support. In addition, communication between employees and managers is improved because they are all actively involved in a unified approach toward improved customer satisfaction.

The second TQC program involves surveying customers in order to define needs as well as gauge their satisfaction with responses to their calls. The survey will be implemented by managment personnel performing telephone interviews of customers. Aside from the obvious outcome of identifying ways the RC's service may be improved, the survey will raise management communication with their customers.

Both projects stress a methodical approach to problem identification and resolution, management support and an effort to meet the customer's needs better. The contrast between the two highlight the flexibility of TQC in approaching various situations.

E. System Support Team.

In addition to the above, the RCs also initiated the System Support Team (SST). This team is dedicated to providing internal support for the PICs engineers. They do not take calls directly but rather act as consultants for the engineers who are actually taking calls. This function is staffed from the Support Engineering organization. By focusing on technical and analytical skills coaching, they can become very proficient in their respective areas of expertise.

Support Engineering in the Response Centers is chartered to provide a high level of technical assistance and training to other elements of the RC and directly to customers as required. In addition, Support Engineering participates in definition and development of new RC support services and productivity tools by working closely with other RC groups. Its mission is:

Provide technical and analytical skills consulting for RCEs.

Assist in the Field Notification process by screening system calls that are candidates for the Field Notification process.

Provide dump analysis service to PICs teams and the System Interrupt team as necessary.

Assist RCEs in down loading patches to customer systems and provide total patch installation to the System Interrupt team.

F. System Interrupt Team.

Another new program implemented recently is the System Interrupt Team (SIT). This team is responsible for taking all system interrupt calls in each RC. The team is dedicated to this function and takes no other calls. The team is jointly staffed by the Support Engineering and Support Operation organizations. By focusing on handling all system interrupts, significant expertise is gained by the team members and subsequently provided to customers.

The mission of the SIT is to:

Provide a group of engineers focusing on assisting customers to restart their systems from a system interrupt expeditiously and in a manner that will maximize data integrity.

Increase customer satisfaction by presenting a consistent methodology that exhibits concern for the customers situation, minimizes down time and provides the best possibility of preventing another interrupt.

Utilize information recorded in TRAKII to consult with the customer regarding any patterns of interrupts that may be forming and keep team members informed of customers who are experiencing multiple failures.

Assure that all appropriate tools are utilized to provide the customer with the most complete information possible about the cause of and any potential remedy to the interrupt.

Increase the productivity of the PICs operation by eliminating high priority interrupt-type calls into one group thereby decreasing engineer 'context switch' time for the remainder of the operations team.

G. Application Notes

Application Notes are intended to provide Hewlett-Packard customers with information in a format that will assist them in using and managing the hardware and software capabilities of their systems. Application Notes are created as a result of analysis of telephone inquiries to the Response Centers where the volume of calls we receive indicates a need for addition to or consolidation of information available through Hewlett-Packard support services.

The Application Notes are distributed with the Software Status Bulletin (SSB) to assure wide availability, but aren't included with every SSB since their creation is determined by the circumstances outlined above.

HP3000 Application Note #1 is the "HP3000 PRINTER CONFIGURATION GUIDE" for HP-IB systems (this excludes the Series II and III.). Hewlett Packard introduced a number of printers in recent years for text, graphic and word-processing applications. This guide covers some of the most common questions that System Managers have in configuring printers on their systems.

HP1000 Application Note #1 is entitled "UPDATING RTE-A - VERSION A.84 TO A.85". This guide contains a description of the steps necessary to update to this release of RTE, A.85, and provides hints to allow one to make a smooth transition to this release.

H. Product division feedback

In addition to continued focus upon our relationship with customers and HP's field organization, we began to look for opportunities to leverage the data we had been collecting for feedback to HP's product development divisions. This required us to examine the data we were collecting and to discuss with the divisions the nature of data they would like to see from us. As a result of We have met with the managment teams of several product divisions. These meetings have been received with enthusiasm and results. The divisions have now found the single source of real customer data they have been looking for and have responded with appropriate zeal. In most cases, these high level meetings have been followed by meetings between RC and division engineers to discuss problem areas in more detail. In some cases, the R&D organizations have sent engineers to actually sit in the RC and take customer calls as a means to provide 'real world' experience. Where enough time has elapsed to see change, we have seen improvement in product quality. This activity continues as we work with the divisions to further refine the type of data we will collect and report back to them. We view this as a major contribution to HP and HP's customers!

I. Future RC Tools

A development team, dedicated to improving customer support through RC tools, is a major advantage to Hewlett Packard's RCs and customers. This team examines the present and future needs of the RCs and then develops and implements the appropriate tools. This means that support cost can be minimized by providing the tools necessary to increase the productivity of RCEs rather than continuing to hire more engineers to keep pace with product introductions and sales. "Working smarter rather than harder" is the phrase we like to use to describe how we can continue to be successful through the use of our own development team. In addition to increasing productivity, these tools also increase the availability of knowledge--knowledge through efficient linking of previous solutions as well as easier to use knowledge gathering ability.

HP's support development team will play an important role in providing superior support and maximizing our customer's return on support dollar investment.

IV. Summary

HP's RCs have grown considerably since they started 17 months ago. They have become an important element of HP's strategy to deliver hardware and software support more effectively. The RCs provide software telephone assistance for all of HP's contractual support customers as well as provide remote diagnosis for HP 3000 hardware calls. There are two RCs in the U.S.--Santa Clara, California and Atlanta, Georgia, which resolve over 3000 system calls per week.

In Europe, HP has recently established Country Response Centers (CRCs) providing software telephone assistance and hardware remote diagnosis. In addition, an operation in Pinewood, England, called Computer Support Pinewood (CSP), has been established as a resource for the CRCs. CSP has a direct line to the Santa Clara RC which enables rapid updating of the information databases for both North American and European customers.

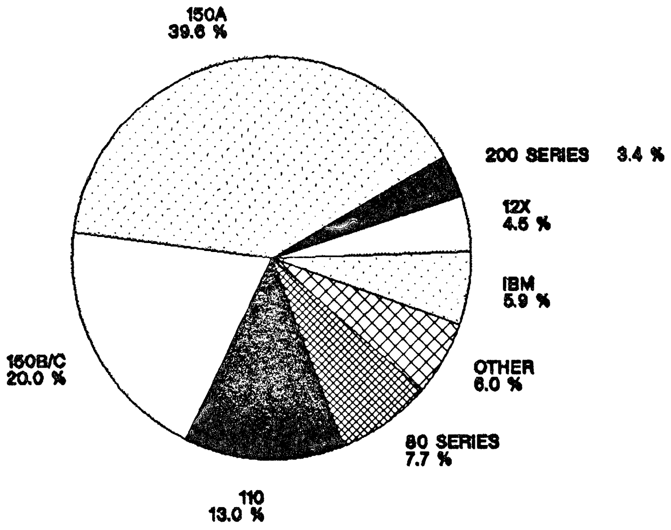
Response time to software questions has been consistently within the two hour specified time usually getting back within an hour. And for hardware problems, the RCs have been particularly successful in initiating problem diagnosis within 30 minutes after a customer places a hardware service call to the local area office. RCEs are able to resolve 95% of the software calls and 20% of the hardware calls without onsite assistance.

The Centers have also been designed to handle their own emergencies. The North American Response Center in Santa Clara has a fully redundant HP3000 Series 68 system to allow complete switch-over within 15 minutes. An uninteruptable power system including an array of batteries and a diesel generator provides continuous backup if a full power outage should occur. Physical security has been addressed with a sophisticated identification system. HP's focal point for Europe, Computer Support Pinewood, has similar backup capabilities to maintain its operation. HP's RCs are often able to back each other up also. For example, at one point a tornado knocked down power lines near the Atlanta Response Center and HP was able to switch the phone lines to the Santa Clara Response Center and continue serving customers until Atlanta resumed operation the next morning.

RCEs have access to information data bases for history on previous customer difficulties as well as a database containing solutions to known problems. Besides the computerized pool of knowledge, the RCs are equipped with more than a dozen different HP systems. Support analysts, sitting at desktop terminals, can reproduce a customer's difficult problem on a duplicate machine or, through a state-of-the-art telephone system, a customer's system. HP engineers can then do even more sophisticated troubleshooting from afar.

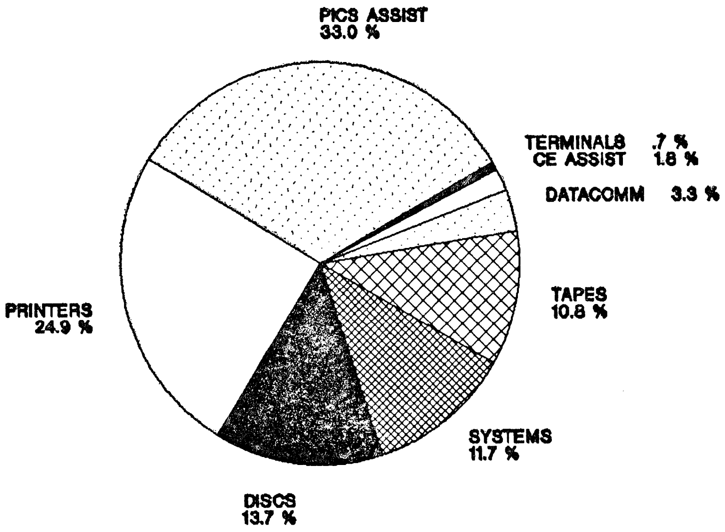
V. Statistical Information

The distribution of PC HELPLINE calls (about 185,000) are:



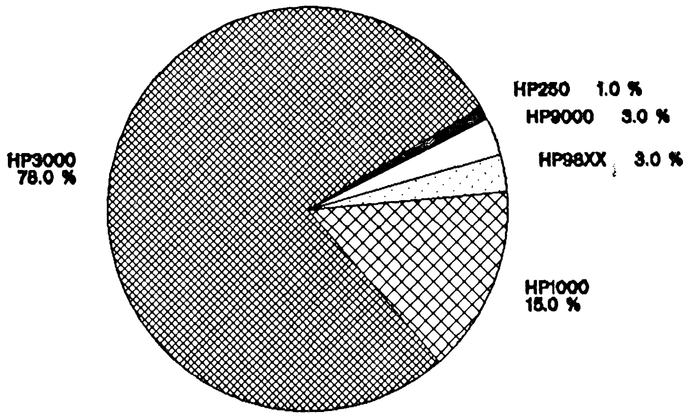
TYPICAL PC CALL SUMMARY BY SYSTEM TYPE

The hardware activity distribution (about 35,000 calls) are:

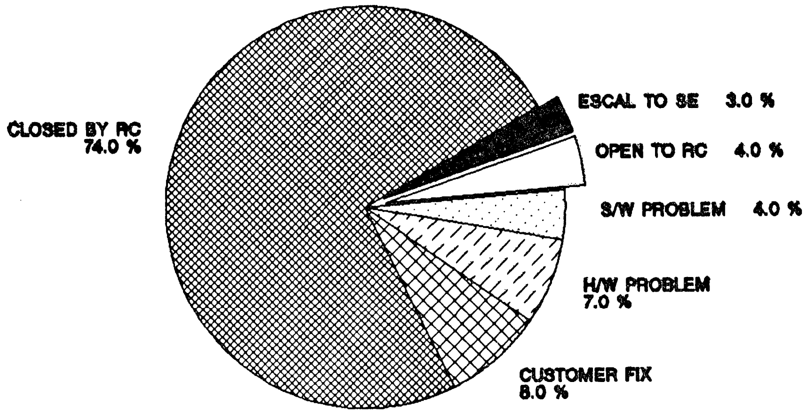


TYPICAL REMOTE SUPPORT CALL DISTRIBUTION

The distribution and resolution of PICs calls (125,000 in North America and 70,000 in Europe) are as follows:



TYPICAL PICS CALL SUMMARY BY SYSTEM TYPE



TYPICAL PICS CALL DISPOSITION

3065. HOW DISPATCHING QUEUES REALLY WORK

Dave Beasley
Computer Systems Division, Hewlett Packard
Cupertino, California

SUMMARY

Have you ever wondered how the MPE scheduler decides which process should be given the CPU? Have you ever wanted to know why some batch jobs seem to have a disastrous effect on on-line response time? If so, you're not alone. This topic seems to emerge as one of the most debated subjects at every user group meeting that I've attended over the past several years. I think that the confusion about how the dispatcher really works is largely due to the fact that the internal algorithms and policies of the dispatcher/scheduler are not well documented. Based on my experience of working daily with the MPE operating system, focusing primarily on the kernel and I/O system, I would like to share with you what I have learned about this subject over the past few years. This paper will focus primarily on how the dispatcher and interrupt system interact and how the scheduling algorithm works. The performance effects of certain events, such as the execution of batch jobs, resource locking, (SIRS and control blocks), and disc I/O will be mentioned where appropriate.

THE DISPATCHER/SCHEDULER

The MPE dispatcher/scheduler, together with the interrupt system, comprise the heart and soul of the MPE kernel (note that the terms dispatcher and scheduler will be used interchangeably). The dispatcher's major responsibilities include selecting the next process to run, invoking the memory manager when a process needs memory, and keeping track of the CPU time used by each process (rescheduling and adjusting priorities accordingly). The dispatcher also invokes the memory management garbage collection routines when the system would otherwise be idle AND when that action is desireable. The dispatcher is invoked explicitly by the DISP machine instruction and runs on a special stack called the Interrupt Control Stack, (ICS). If a DISP instruction is executed on the ICS (ie., an interrupt handler is running), the dispatcher is not entered immediately, but is deferred by setting a "dispatch requested" flag. This is to allow all interrupt handlers executing on the ICS to complete (note that we could have stacked interrupts due to one interrupt interrupting another). Each interrupt handler that executes on the ICS must do an IXIT instruction, (a special type of EXIT), when it has finished processing. When all interrupts have been processed, the last IXIT executed must decide whether or not to enter the dispatcher code by "EXIT'ing" through a special "dispatcher stack marker", or to return to whatever was interrupted.

When no process is running, the CPU is either executing in an interrupt handler and the I/O completion routines that it may call, in the dispatcher, or in the memory management routines, (called by the dispatcher). Before proceeding with the algorithm and policies of the scheduler, we must define what a process actually consists of and discuss some key points about the interrupt system.

WHAT IS A PROCESS?

As the well informed HP3000 user, which I'm sure you are, you must have read that "a process is the unique execution of a program at any given time". This statement is absolutely correct, but I'm going to expound on this definition based on the outside chance that even one person reading this may be a little confused. Each time that you logon, a Command Interpreter process is created for you by the system. Each time that you :RUN a program, a new process is created. If an executing process CREATES and ACTIVATES a son program, a new and distinct process is created. Each process on the system has some unique attributes. Each one has its own unique stack for its own private copy of the program's data. Each process has one or more code segments associated with it. Since the HP3000 allows code sharing, these code segments may be shared between processes running the same program or sharing SL segments. Each process may also have one or more extra data segments associated with it, some of which may be needed by MPE for system tables and file control blocks, and possibly some that are acquired by the program for its own use. Each process is also given an entry in the Process Control Block Table, (PCB). The PCB table contains all of the status and necessary information that the dispatcher needs to decide which process should be given the CPU next. Each PCB entry contains critical information about the process. For example, the PCB entry informs the dispatcher whether the process is ready to run and only needs the CPU, or if the process is blocked, waiting for some event to occur. Some of the events that a process could be waiting on are I/O waits, memory waits, or critical resource waits, (ie., a SIR or file control block wait). It is the job of the dispatcher to schedule processes to run based on the information found in the PCB table.

THE INTERRUPT SYSTEM

It is well beyond the scope of this paper to fully explore the interrupt system, however it is essential that you understand some of the key points. MPE is an interrupt driven operating system, meaning that MPE does not "go around looking" for things to do, but rather is informed of pending activity and/or the completion of certain events by way of an interrupt from some device or a "message" from some part of the operating system. There are two basic types of interrupts on the HP3000: internal and external. Internal interrupts are a result of the CPU and its microcode detecting some unusual or abnormal condition. Some examples of internal interrupts are stack overflows, absence traps, (referencing a segment that is not in memory), integer overflows, and bounds violations. There are many others. In most cases, these internal interrupts are handled directly on the user's stack, but some of the interrupts must be handled on the ICS. Additionally, in certain instances, the dispatcher must be invoked on behalf of the process to perform some function, such as initiating and handling a stack overflow or absence trap. The second type of interrupt is called an external interrupt. These can be thought of as device interrupts. External interrupts are always processed by their interrupt handlers on the ICS. External interrupts usually indicate a change in the status of an I/O device, such as a magnetic tape being placed on-line, or the completion of an I/O operation, such as a disc read or write. The external interrupt handlers and the I/O completion procedures that they typically call, may call the procedure AWAKE to signify that some event has completed. AWAKE will indicate in the appropriate PCB entry that the event has happened; if it was the event for which the process was waiting, then AWAKE will request that the process be inserted into the dispatcher's list of processes waiting for the CPU. AWAKE may or may not do a DISP instruction at this time. We will discuss AWAKE's role in process preemption and in communication with the dispatcher later.

POLICIES OF THE MPE SCHEDULER

All decisions about CPU scheduling are made by the scheduler. The scheduler maintains a DISPATCH queue of all processes in the PCB table that are READY to run. Being READY to run implies that a process does not need to wait for some event to occur or for some system resource to become available before it is able to execute. All that a READY process needs is to have the CPU allocated to it. Now that I've said this, there are two exceptions to the previous statement about READY processes on the DISPATCH queue. A process that is "short waited" for disc I/O is not taken off of the DISPATCH queue since those waits are usually satisfied so quickly that it is not worth the effort to take them off. Of course, they will not be launched until the I/O is complete. The other exception is that processes waiting on memory resources are also on the DISPATCH queue, since the dispatcher is responsible for initiating memory management activity. A process which becomes READY, due to the completion of some event or some resource becoming available, is inserted into the DISPATCH queue in priority order. The scheduler will ALWAYS ATTEMPT to allocate the CPU to the highest priority process that is READY to run. Under certain circumstances, a higher priority process may become READY while a lower priority process is executing. This can occur when an executing process is interrupted, and the reason for the interrupt satisfies the "wait reason" for a higher priority process. After AWAKE requests that the process be inserted into the DISPATCH queue, it must determine if it should take the CPU away from the current process in order to allow the dispatcher to run the higher priority process, or if it should allow the lower priority process to run until it blocks naturally. There are some specific rules that AWAKE must follow with respect to process switching, which we will discuss later. The point here is that if the process switch does not occur, this is the only time that the highest priority process does not run, and it's a very temporary condition.

WHEN DOES A PROCESS GIVE UP THE CPU?

A process will run until it blocks naturally, is preempted by a higher priority process becoming ready for the CPU, or until the dispatcher decides that the process has had the CPU long enough.

BLOCKING NATURALLY

To block naturally simply means that the process requested some resource that wasn't available, such as a SIR, a lock on a file control block, a system buffer,etc. A process will also block naturally when it has requested I/O or if it needs memory resources. A process requests to be blocked by calling the procedure WAIT, specifying what type of event it expects to occur before it can execute again. WAIT stores this reason in the PCB entry for the process. When the procedure AWAKE is called to awaken a process for a particular event, it will update the PCB entry to indicate that the event has occured. If the event corresponds to the reason why the process was waiting, AWAKE will request that the process be placed in the DISPATCH queue of READY processes.

PROCESS PREEMPTION

A process is said to be preempted when the CPU is taken away from it, before it blocks naturally, in order to allocate the CPU to a higher priority process. After verifying that the reason the new process is being awakened corresponds to the reason that the process was originally blocked, AWAKE requests that the process be inserted into the DISPATCH queue in priority order. If AWAKE determines that the new process being awakened is of higher priority than the one that has been interrupted, AND IF the dispatcher has enabled process preemption for that process, AWAKE will do a DISP to initiate the process switch. We will discuss the rules for process preemption and the role of the procedure AWAKE in this effort after we discuss the characteristics of the scheduling queues.

HOW DOES THE DISPATCHER DECIDE HOW MUCH IS TOO MUCH?

Every 100 milliseconds, the system clock generates an interrupt, which causes control of the CPU to be transferred from the executing process to the ICS, where the interrupt handler for the system clock will execute. Each time that a process is launched, it is given three "ticks". In other words, a variable, which we'll call QTIME, gets set to 3, representing 300 milliseconds of CPU time. Each time that the clock interrupt handler executes, it decrements QTIME. Whenever QTIME falls to zero, a DISP instruction is executed to invoke the dispatcher. The dispatcher will then treat this process just as if it had voluntarily given up the CPU or had been preempted. The dispatcher will reschedule this process, adjusting its priority if necessary, and will select the highest priority process on the DISPATCH queue to launch. Note that the new highest priority process may be the same one that the clock interrupt handler preempted because QTIME was zero. Many people still confuse the above method of allowing the clock interrupt handler to preempt a process with the idea of the process using a "quantum". In fact, under the MPE III operating system, we did call it that! There was even a :QUANTUM command that allowed the system manager to determine how many "ticks" or milliseconds a process would get as its "quantum". However, it is important to note that this method is not used to determine a "quantum" in MPE IV and MPE V! The dispatcher uses different criteria to determine if the process has used more than its "fair share" of the CPU, (or has exceeded its "filter" value). Preempting a process because it has used 300 milliseconds of consecutive CPU time is only done to allow the scheduler to reschedule this CPU intensive process to ensure that it does not exceed the "filter value" for its scheduling queue.

SCHEDULING QUEUE CHARACTERISTICS

There are five distinct scheduling subqueues in MPE. The AS and BS queues are considered to be linear queues, meaning that the scheduling algorithm makes no priority adjustments to processes in these subqueues. The CS, DS, and ES subqueues have always been considered to be "circular" queues. This term originated because of the way that priorities of processes in these queues gradually moved up and down, resembling a "circular" shape, in MPE III. This is not the case in MPE IV and MPE V, however much of the documentation still refers to these queues as "circular" queues, (if you must make a geometric analogy, they probably appear more like "semicircles"). The scheduler controls the priority of processes in these queues.

The scheduling queue priorities range from 0 to 255, with 255 being the lowest priority. When a process is created, it is scheduled by default into the same scheduling queue as its father. However, the scheduling queue and/or the absolute priority number of the new process may be specified in the CREATE, CREATEPROCESS, or GETPRIORITY intrinsics.

AS AND BS SUBQUEUES

The priority range for the AS subqueue is from 0 to 100. This queue is typically reserved for high priority MPE processes. The priority range for the BS subqueue is from 101 to 149. Some MPE processes run in this queue also. This queue may also be used by high priority user processes if the user specifies. With the proper capabilities, a user can specify any priority for any process. It should be pointed out, however, that it is possible to deadlock the system or to adversely effect system performance by using these subqueues, particularly the AS queue. Once a priority has been assigned to a process in one of these linear queues, the scheduling algorithm does not alter it.

CS SUBQUEUE

The strategy of the CS subqueue scheduling algorithm attempts to favor interactive users. The default range of priorities in this queue is from 152, which is called CBASE, to 202, which is called CLIMIT. These values may be altered by the :TUNE command. A process' priority is dynamic within this subqueue and is controlled by the scheduler. Whenever a process blocks for a terminal read, it is rescheduled at CBASE priority. Therefore, the first time that a process runs after being awakened from a terminal read, it is launched at CBASE priority. A process may use the CPU and then block, or give it up, several times before it blocks on another terminal read. For example, a process may block for several disc I/O's or it may be impeded for some system resource before it requests another terminal read. We call the CPU time used between terminal reads, a "transaction". A process is rescheduled each time that it gives up the CPU, (or quiesces). Its priority may be decremented, (incremented numerically), if it exceeds the filter value for the CS subqueue, or if it blocks due to a memory trap. In MPE V, its priority is not decremented due to a memory trap, but only for exceeding the filter value. Additionally, a process in the CS queue is enabled for preemption if it exceeds the filter value. Once the priority of a process is decremented, it is never incremented again until it blocks for a terminal read, (which completes a "transaction"), and it is assigned the CBASE priority at that time. A process' priority may never be decremented below the CLIMIT priority. There is one exception to the scheduling algorithm which may cause a process' priority to be raised without having blocked for a terminal read. We will discuss this exception after the discussion of each of the scheduling queues. How does the scheduler determine the filter value for the CS subqueue? The filter value is also called the "average short transaction" time, or AST. The AST is a computed average of the time required for all "transactions" to complete for every interactive process on the system. Remember that a "transaction" is defined to be the amount of CPU time used between terminal reads. Each time that a process blocks, or is preempted, the scheduler keeps track of the CPU time used since the last time the process was awakened from a terminal read, therefore, when the process blocks for a terminal read again, the scheduler knows the time used for the current transaction. A formula is then used to calculate a weighted AST which is effected by each interactive process on the system. Processes in the DS and ES queues which are running interactively, and therefore complete transactions, also contribute to the AST, even though the AST value is not used as the filter value for processes in the DS and

ES queues. The AST value will never be allowed to exceed the CQ MINQUANTUM and CS MAXQUANTUM limits, which are set by default to 0 and 300 milliseconds respectively, or which may be set by the :TUNE command.

DS AND ES SUBQUEUES

The DS and ES subqueues are primarily intended for batch jobs, although sessions may run in these queues if so desired. The default range of priorities for the DS queue is from 202, which is called DBASE, to 248, which is called DLIMIT. The default EBASE value is 250, and the default ELIMIT value is 255. These values may be altered by the :TUNE command. When processes are created in either of these subqueues, they are originally placed on the dispatch queue at their BASE priority. Each time that a process in one of these queues blocks for any reason, if the total CPU time accumulated exceeds the "background filter", its priority is decremented, (numerically incremented by one). In MPE IV, its priority is also decremented, (numerically incremented by one), if it blocks for a memory absence trap, (this does not happen in MPE V). A process' priority will never go below the LIMIT value for its scheduling queue. When a process is not executing interactively, such as a job, once its priority has been lowered it will not be raised again by the scheduling algorithm. Again, the same exception that applies to the other queues exists for the DS and ES queues also. We will discuss the exception shortly. If a process is executing interactively, and it is awakened from a terminal read, it will be placed on the dispatch queue at the BASE priority for its scheduling queue.

How is the "background filter" determined for the DS and ES subqueues? Although the :TUNE command allows you to specify a MINQUANTUM and MAXQUANTUM value to be used as the filter values for these queues, the filter value is constant and is the value specified as the MAXQUANTUM for the DS queue.

THE BIG EXCEPTION

There is one exception to all of the above rules for scheduling processes. A process in any subqueue may have its priority temporarily raised if it is the holder of certain critical resources and another higher priority process wants one or more of those resources. The only two resources for which this exception applies are SIR's and IMAGE data base control blocks. A SIR, (System Internal Resource), is a semaphore that is used to guard against unwanted concurrent access to certain system resources, such as system tables or the system directory. If one process is the holder of a particular SIR that another process also wants, the second process must wait until the first process releases the SIR. In MPE IV, this "SIR queue" is priority based. For example, if a higher priority process queues up for a SIR which is held by a lower priority process, the lower priority process is given "SIR priority". In other words, its priority is temporarily raised to that of the higher priority process. As soon as the SIR is released, the higher priority process will be given the SIR and the process that was given "SIR priority" is returned to its original priority. Intervening processes in the "SIR queue" will remained queued, but the higher priority process will take precedance in gaining access to the SIR. In MPE V, the "SIR queue" has been changed to a FIFO based queue. If a higher priority process queues for a SIR which is held by a lower priority process, then the process holding the SIR and all intervening processes queued for the SIR will be given "SIR priority". The SIR will then be given to each of the processes in the order that they queued for it. If a process is the holder of an IMAGE data base control block, and a higher priority process requests it, the holder and all intervening processes that are queued for the control block have their priorities raised to that of the higher priority process requesting the control block. Once each process has released the resource, its priority is lowered back to its original priority. Note that as of MPE V/E, processes in the CS, DS, or ES queues, will not have their priorities raised above the CBASE priority due to impeding a higher priority process for an image control block. There are many other kinds of resources that processes may queue up for. Some examples of these are RINS, locks for file system control blocks, message buffers, IOQ and DRQ entries if too few are configured, and many more. Note that higher priority processes simply have to wait behind lower priority processes for these types of resources.

WHAT DOES AWAKE DO WHEN A PROCESS BECOMES READY?

If the dispatcher is running and is in PAUSE, meaning that there are no processes currently needing the CPU, AWAKE will do a DISP instruction to restart the dispatcher. The dispatcher will then scan the list of READY processes and launch the one with the highest priority, (which would in this case be the one we have just awakened). If the dispatcher is running, but is not in PAUSE, it is either preparing a process to launch, it has called the memory management routines on behalf of some process, or it is calling memory garbage collection routines. If this is the case, AWAKE simply places the priority of the process it is trying to awaken in a special memory location that is reserved for communication between AWAKE and the dispatcher. The dispatcher will look at this location at convenient points to determine if there is a more urgent process to launch than the activity that is currently in progress. If the dispatcher is not running, it means that a process is currently running. Actually, it means that we have interrupted the last process launched by the dispatcher. AWAKE must decide whether or not to preempt this process and initiate a process switch, or to allow the executing process to run until it blocks naturally. The following rules apply to process preemption.

1) Any process in a linear queue may be preempted by any process that has a higher priority, regardless of the priority of the scheduling queue that it is in.

2) Any process in any queue may be preempted by a higher priority process that is also in a higher priority scheduling queue. For example, the CS queue has a higher priority than the DS queue which is higher than the ES queue.

3) A process in the CS queue may be preempted by a higher priority process in the CS, DS, or ES queue if and only if it has been enabled for preemption. The dispatcher enables CS processes for preemption only when they have exceeded the AST, (average short transaction time).

4) Processes in the DS and ES queues are never enabled for preemption which means that they can never be preempted by processes in their same queue, regardless of their priority.

PRIORITIZED DISC I/O

Since the introduction of MPE IV, disc I/O has been prioritized. If a process issues a disc request, the disc request is issued at the priority of the process requesting the I/O. Memory management I/O is also issued at the priority of the process that needs the memory. Background writes, or anticipatory writes, are issued by the memory manager at a background priority, which is 255. Background writes are issued in order to update the copy of a data segment on disc in anticipation of having to swap out the data segment. Since the segment may be "recovered" in memory, therefore invalidating the need for the disc write, these types of writes are performed in the background, so as not to interfere with useful I/O. If the memory manager decides that the memory space is needed, and if the background write has not completed, then the priority of the disc request for the background write is bumped up to the priority of the process that needs the memory space.

PERFORMANCE IMPLICATIONS OF BATCHJOBS

There are several characteristics of the scheduling algorithm and scheduling queues which are quite good at attempting to ensure that batchjobs really do execute in the background. (Although, with the :TUNE command, you can arrange the scheduling queue limits to allow jobs to compete with sessions, or to even be favored). One characteristic in particular is that the scheduler allows higher priority processes in higher priórity scheduling queues to preempt lower priority processes as soon as they become ready for the CPU. For example, while one process in the CS queue is waiting for terminal I/O, a CPU intensive batch job in the DS queue may be executing. However, as soon as the terminal read is completed, the higher priority CS queue process will preempt the DS queue process, thus favoring the interactive user. Another mechanism which prevents CPU intensive processes from adversely effecting system performance is the system clock interrupt handler, which prevents a process from using more than 300 milliseconds of CPU time consecutively without being rescheduled. Another characteristic which favors the interactive user is prioritized disc I/O, which tends to allow I/O from interactive sessions to take precedence over I/O from lower priority batch processes. However, in spite of the attempts to ensure that sessions are favored over batch jobs in scheduling, low priority processes that compete for the same resources as higher priority processes can seriously effect performance on busy systems.

Once a batch job, or low priority process, acquires a critical resource that is needed by a higher priority process, the higher priority process must simply wait until the resource is released. If there are a large number of processes on the dispatch queue ahead of the lower priority process, obviously it cannot run in order to release the resource. To complicate the scenario, if the low priority process has outstanding disc I/O, and if there is a significant amount of I/O being issued at higher priorities, the low priority process that holds the resource may have to wait a considerable amount of time for the I/O to complete. In the mean time, the higher priority process that wants the resource will simply have to wait, as well as any other process that queues for the resource. As we have previously discussed, an attempt was made by the scheduling algorithm to give low priority processes a higher priority, temporarily, when they are impeding higher priority processes. This was only done for two resources, however; SIR's and IMAGE control blocks. Although this works well in most cases, if the lower priority process had previously issued disc I/O which was not complete, the priority of the outstanding disc I/O is not bumped up in priority. Therefore, if a considerable amount of disc I/O is being queued up at a higher priority, the I/O request for the lower priority process that holds the resource may be delayed, in spite of the fact that its CPU priority had been temporarily increased.

CONCLUSION

As you can plainly see, there are many variables in determining how one or more batch jobs will effect the performance of a system. The most obvious ones are the total number of processes on the system and the scheduling queues that they execute in, and the amount of disc I/O that is queued up at any one time. What is not so obvious is what effect competition for the same resources, by sessions and batch jobs, have on the scheduling algorithm.

I hope that this paper has helped you understand this to some degree.

It is almost impossible to recommend "rules" and "guidelines" about what is the ideal mix of batch jobs and sessions. The only recommendation that I will make is that you attempt to minimize resource competition between jobs and sessions. One way in which this can be done is by minimizing, or not running, jobs during peak hours that utilize the same files and data bases as online users. Of course, it will be impossible to eliminate competition unless you simply defer all jobs to off hours. This assumes that you have "off hours" in your shop! Good luck and happy tuning!

Dave Beasley is a Systems Support Engineer for Computer Systems Division of Hewlett Packard in Cupertino, California. Dave has worked for Hewlett Packard for 6 years and has been a user of HP3000's for 8 years.

3066. An Information Crossroads Decision: A Blend of HP3000 and HP9000 for Computer Graphics

Sam Boles, Member Technical Staff
Hewlett-Packard

. . . a picture's worth a thousand words . . . and on a computer can take more processing and storage than 10,000 words . . . here are some tips on performance, resolution and load balancing that can help you with . . .

We all know that a picture's worth a thousand words.

And those of us who've drawn pictures with a computer know a picture can take more processing and storage than 10,000 words.

Then once you've gotten the basics under your belt, you get harder to please. You want resolution. You want performance. You want those end-points to meet, you want your curves smoother . . . you might even want animation . . . but, for sure, you don't want to wait.

This growing colony of computer artists with their growing appetite for the artistic can bring a multi-user commercial machine to its knees . . . unless you resort to second-order Distributed Systems.

You've experienced first-order DS with your HP3000 talking to others like it, sharing programs and databases. There's a second-order distribution that networks into your HP3000 the high-performance high-resolution graphics capabilities of the HP9000. This gives you better pictures faster -- and without degrading the performance of your main-line transaction processing applications that make the profits that you wanted to draw the pictures about in the first place. And you can still use the data-words-graphics integration of your HP3000 with its laser printers and other powerful peripherals.

The war stories of the extensive graphics and technical publication requirements of the HP Systems Productivity Center may give you some productivity ideas for your own installation.

. . . why graphics? . . .

for those who can't

read . . .

Once upon a time, there was a magnificent piece of americana called Life. It came out once a week. And it cost only a dime. It was the week that was, in pictures. The glory of victory, the agony of defeat. The blood, sweat and tears that started wars and finished wars. The laughter, the sobs, the bad, beautiful, noble and ludicrous of the human condition -- photographed by some of the most courageous men and women in the history of journalism.

But not everyone viewed this piece of americana the same way. A young undergraduate (an English major) once remarked,

"Life is for those who can't read . . .

Time, for those who can't think . . . ."

Without commenting on its validity let's see if we can leverage this wisdom of a generation past, and come up with an answer to the question

Why Graphics?

Graphics is for those who can't read. No, it's not that they can't read because they can't read. They can't read because they don't have time to read.

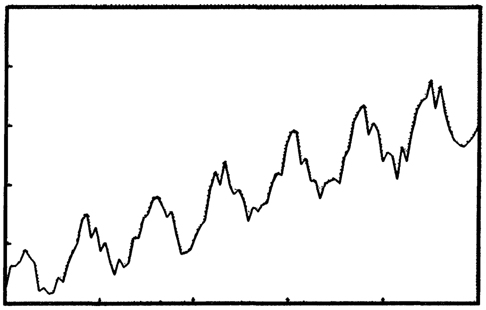
Let's look at an example out of one of the HP9000 reference manuals. Read these numbers:

```
0.1610 0.1625 0.1625 0.1628 0.1636 0.1631 0.1627 0.1608
0.1610 0.1606 0.1607 0.1617 0.1614 0.1626 0.1634 0.1640
0.1656 0.1660 0.1644 0.1651 0.1635 0.1641 0.1628 0.1619
0.1630 0.1624 0.1627 0.1644 0.1644 0.1657 0.1660 0.1670
0.1672 0.1666 0.1658 0.1662 0.1646 0.1633 0.1634 0.1636
0.1645 0.1652 0.1656 0.1677 0.1689 0.1680 0.1696 0.1680
0.1674 0.1677 0.1669 0.1655 0.1665 0.1662 0.1667 0.1668
0.1681 0.1688 0.1687 0.1707 0.1716 0.1716 0.1694 0.1698
0.1683 0.1683 0.1671 0.1681 0.1683 0.1684 0.1681 0.1698
0.1705 0.1723 0.1730 0.1734 0.1714 0.1722 0.1716 0.1696
0.1702 0.1699 0.1684 0.1706 0.1696 0.1715 0.1730 0.1737
0.1739 0.1751 0.1732 0.1747 0.1729 0.1717 0.1710 0.1707
0.1706 0.1709 0.1713 0.1720
```

Did you read them? No, of course you didn't. You don't have time to read . . . really read . . . a hundred numbers. And if you did have time you wouldn't waste it like that. (Notice how boring it gets after about the third digit?)

Besides why read a hundred numbers when today's technology can read them for you and maybe tell you something you might have missed -- because it's in between the lines. Just browsing probably gives you the overall trend, but how about periodic motions and number of cycles?

Why Graphics?
The 100 Numbers You Didn't Read



Better, Easier, FASTER!

. . . and for those who

can't think . . . again,

because they don't

have the time . . .

This maybe tells us something else, too: Graphics is for those who can't think. No, it's not that they can't think because they can't think. They can't think because they don't have time to think.

Sure, they could take a pencil and paper and calculate the deltas and get the pattern clusters in a few minutes. But that few minutes has an opportunity cost. What they could've done. Like get the corrective action launched. Or the next step of the design underway. Or whatever is the real work they could've been doing.

. . . to save time . . .

the Ultimate

Unreplenishable . . .

at a rate of a

thousand words per . . .

It's the old story of "a picture's worth a thousand words." You get the message to your audience better . . . faster. You save time. They save time.

Let's look at this thing called time. It's a unique commodity. Or maybe it's not a commodity -- ever try to buy some time? Anyway, it's unique.

Remember back in the early 70's when we had to stand in line to get gas? Geologists for years had been telling us we were burning oil faster than it was being replenished. Then we finally got the message: That meant we could run out. So the oil owners lowered production and raised the price. And suddenly there we were, waiting in line for stuff that not long before that a Gulf station in Los Angeles would sell for 18.9 cents a gallon during a gas war . . . and clean your windshield while you bought it.

All of that for something relatively unreplenishable. But look at time. The Ultimate Unreplenishable.

If oil is slow at replenishing, it's nothing compared to Time. Time doesn't replenish at all. You burn it and it's gone. Forever. And we burn it every second of every minute of every day. Not just when we drive to work. And we burn more of it faster all the time. Ask anyone who's been around awhile about how much faster you burn it as years go by. And how much more of it you burn.

The Ultimate Unreplenishable.

Anything that improves performance in time utilization deserves attention. And graphics is one of those things.

So much for Why Graphics? Let's look at the evolution of the computer graphics artist.

. . . the evolution of the

artist . . . basics under

your belt, you get

harder to please . . .

Remember a few years ago when DSG (Decision Support Graphics) brought charts right to our terminal on the HP3000? Bar charts, line charts, pie charts. We could eavesdrop a plotter on the line and get hard copy on the spot in minutes. Then there was HPDRAW to build the text and picture slides to go with the charts.

We could build our slide presentations quickly and conveniently. And update with the latest numbers in a matter of minutes.

The magic of Interface Reduction.

We no longer had to queue up at the graphics department. And wait for the typesetter. We reduced these interfaces to zero. We saved time and money. A quantum leap in Productivity thru Interface Reduction.

It was great.

Then as the elation wore off we noticed you could tell the computer slides from the typeset slides. It was the letters. Those stick letters. Like tapioca: good, but not exciting.

We needed better letters. Nice smooth spline curves, with fill and boundary in different colors. And a wider range of fonts and sizes.

So DRAW II arrived with really world-class letters. A quantum leap in professional quality lettering.

It was great.

Then as the elation wore off we noticed that next to the beautiful spline letters you really noticed when the end points in our drawings didn't quite meet, and some of the detail was a little ragged.

We needed better resolution, and better and faster zoom, pan, grid snapping and . . .

On top of all that, something was happening in our work area.

Our colleagues were trying to figure out how we were able to get such good presentations so fast and still come in under budget. They saw we were using DSG and DRAW, and they started to do the same thing.

CPU utilization began to grow. The last 3 days before quarterly review, response time got really slow. We upgraded to a 68, and that helped. But not enough to keep up with the growing popularity of the tool.

. . . the issues of resolution, response time and load balancing . . .

As we experienced more and more positive results from our graphics, we wanted more and more quantity and quality.

More people were starting to use the tool. And the positive results from using the tool made them use it more.

Each iteration made the users more proficient with the tool, enabling them to do a better job of the next slide and increasing their appetite for perfection proportionately.

The need for fast-response high-resolution graphics tools evolved as the degradation of response time evolved, aggravating the economic imbalance with a diminishing supply of CPU cycles, disc I/Os and main memory being confronted with an increasing demand for a higher service level by more users.

Compound the situation with a 25 per cent annual growth rate and you get the scene that led us into second-order Distributed Systems for our computer graphics.

We already had our 3000's DS-linked so we could get at programs and databases on neighboring machines. And if we had a heavy-duty crunch that we needed to run we could off-load this to one of the light-load nodes.

But with the graphics overload we were looking for more than just CPU cycles. We were looking for functionality. A richer command set. A more natural human interface. Higher resolution. Faster graphics performance in first-draws, redraws, transformation of primitives and cells.

We turned to our CAD-CAM family, the HP9000, and found what we were looking for.

Let's look at where the 9000 was coming from.

. . . the HP9000

genealogy . . . a "PC"

before they invented

the word . . .

If you trace the roots of today's HP9000 family, you go back to the days when we used words like "programmable calculator" and "desktop computer." If you look at the HP9825 (circa mid-70's) you see the low-cost, small footprint, portable and individual work station that might've been called a "PC" if we'd used that kind of language in the medieval days of 10 years ago.

When we retired its jersey some time back, the HP9825 had been one of the top unit sellers in the history of Hewlett-Packard.

The tradition evolved into the low-cost HP9000/200 with 8- and 12-Megahertz processors with 3- and 4-plane color graphics, and the HP9000/500 with 3 CPU's in a box supporting 8 planes of color (that's 256 colors from a palette of 16 million) with a graphics accelerator that pumps 60,000 vectors a second over a 2 Megabyte bandwidth bus.

What all that bottom-lines to is high-performance high-resolution graphics with a wide range of price points.

Let's take a look at what the low-cost range of this spread can do for your resolution, performance and load-balancing problems.

. . . a rich

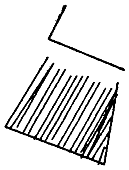
repertoire of

commands, primitives

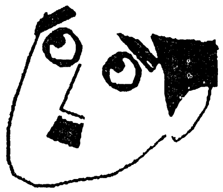
and structurism . . .

Even in the low-cost HP9000/200 you find a feature set with a functional richness that puts you in a new graphics domain.

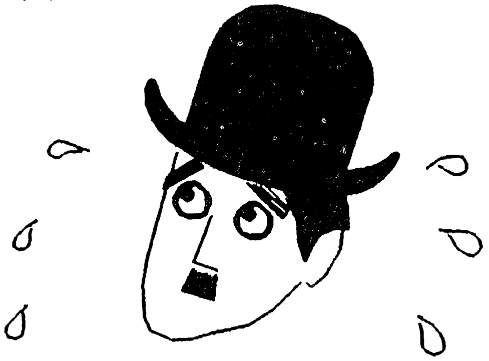
With tools like EGS (Engineering Graphics System), you can zoom in on a particular detail and get the positioning you want, right down to the whiskers on the face (actually, a whisker's about 50 microns in diameter and the system has sub-micron resolution capabilities) . . .



Using the cell instantiation, component and level display selectivity and other functions so essential to CAD applications such as integrated circuit layout, you can build a basic cell one time . . .

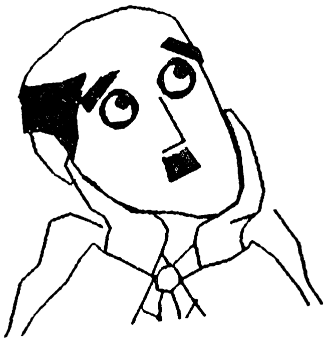


. . . and scale and package it with a given instantiation in a given context . . .



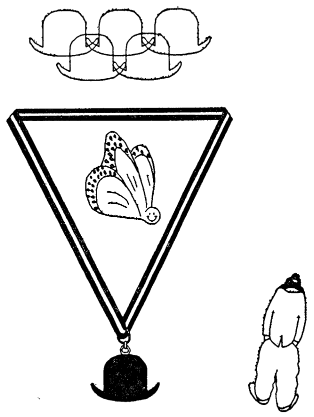
. . . include it in another instantiation with a different context . . .

. . . mirror the same cell for a different orientation in still another context . . .



. . . build up your hierarchical structure of cellular components with whatever scaling, translation, rotation, mirroring, zooming, panning are required, till you have the modules arranged in a multi-level composite that is your complete circuit . . . or whatever it is you're building . . .

. . . you can then take your integrated circuit layout or whatever it is you're building, "plot to disc" so you get an ASCII form of the HPGL commands that normally drive a plotter. You then use a terminal emulator (LAN's on its way, so your 9600 baud can move to the multi-megabit range) to get the vectors from the 9000 to the 3000. Here you've got the full power of TDP, lasers and other technology (such as EGS2FIG in the Contributed Library) to do your final packaging. Once established, your components can go into a library to provide a leverage base for future fan-out . . .



Epilogue . . .

The odyssey spans computer domains, operating systems, design disciplines. It gives you the rich functionality of the HP9000 CAD/CAM world, with it's high resolution, instantaneous response and natural human interface. It gives you the powerful data-words-graphics capabilities of the HP3000 and the laser printer. And it smoothes your processing load by spreading it out across the appropriate nodes to make your general user population a bit happier as their data processing engine is a little more responsive to their touch.

About the Author . . . .

Sam Boles is a Member Technical Staff in the Productivity Center at the Hewlett-Packard computer facility in Cupertino, California. With HP since 1976, his computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on networking HP productivity technology. Sam received his MS at UCLA in Information Systems.

3067. Developing Cost Effective Applications and Utilities Using HP Business BASIC/3000.

Mark L. Hoeft
Language Development Engineer, HP Business BASIC/3000
Computer Languages Lab
19447 Pruneridge Avenue,
Cupertino, California 95014, USA

Abstract:

HP Business BASIC/3000 is a very powerful language which, when used properly, can be a cost effective solution to the serious problem of collecting data and converting it into meaningful information. The wide variety and richness of the features in HP Business BASIC/3000, which accesses several of the subsystems available on the HP3000 through language constructs, make it very easy to write both massive applications and small utility programs quickly. These features allow programmers to concentrate on the problem at hand instead of concentrating on the details of language and subsystems.

In order to make the best use of any tool it is necessary to understand how it works and how to make the best use of its features. This paper will introduce some of features of HP Business BASIC/3000 which make programming more productive and describe how to tune the performance of programs written in this language. Other topics include the intended purpose of HP Business BASIC/3000, the system resource requirements of its features and its portability from/to other systems.

Table of Contents:

1.0 INTRODUCTION

HP Business BASIC/3000 (HPBB) is a language which was recently introduced on the HP3000. Based on BASIC/250, HPBB provides many powerful features besides those found in most programming languages and also an entire environment for developing applications and utilities. These enhanced features and this interactive environment combine to provide very high programmer productivity resulting in cost effective software. HPBB is well suited to easily processing information into a more understandable format. It does a lot of the work that programmers now need to do when accessing several of the subsystems of the HP3000, allowing the programmer to deal with the problem instead of the tool.

HP Business BASIC was started in August 1981, after several years of investigation and specification. It has taken over 50 man-years to complete, and contains over 310,000 lines of PASCAL source (180,000 lines of real code). Its goals were to provide an upgrade path from the HP250 to the HP3000 (both are commercial systems) and to provide a complete, modern BASIC for the HP3000. It it a complete program development system providing editing, debugging and program execution in one environment.

HP Business BASIC/3000 is intended to be used in developing business applications and utilities. Its easy subsystems access makes it convenient to write large applications and its rich language features and interpretive environment make it easy to write quick utilities. It is based on BASIC/250, which has one of the highest user satisfaction ratings in the industry.

This paper will introduce many of the features which increase the productivity of those who program in HP Business BASIC and will describe how to make the applications and utilities they write run faster. The resource requirements of HPBB and HPBB's compatibility with other BASICs will also be discussed.

2.0 DEVELOPING APPLICATIONS AND UTILITIES

One of the biggest problems in developing applications and utilities in the languages currently available has been that these languages have been limited in features specifically aimed for commercial applications. The features supported have typically been control statements and data structures. Some examples of control statements are used for declaring and calling procedures, evaluating expressions and branching (conditionally or unconditionally). If one wanted to perform some complex action, then one would use these building blocks together to perform it. Sometimes subsystems for doing common actions would exist in libraries that could be called from user programs. These libraries provided a lot of flexibility and could be called from a number of languages. As a result it was very difficult for programmers to write programs to use these subsystems. They found that they spend a large amount of time figuring out how to use the subsystem instead implementing what they wanted to do. Also there were a lot of common actions which were not in subsystem libraries and the programmer had to "re-invent the wheel" each time these actions were needed. Needless to say, this had a large negative impact on the productivity of the programmers, and as a result, the time and cost needed to write software. Even a small utility could be difficult to write.

HP Business BASIC/3000 (HPBB) solves many of these problems. It has a wide variety and richness of features, of which a portion will be presented here. The language has many of the desirable features which are present in other languages (but not in the combinations present in HPBB) and they make it easy to access many of the subsystems of the HP3000. HPBB does much of the "house keeping" necessary and does many of the common actions faster that if they are written by the programmer. In general HPBB's features allow the programmer to spend time on WHAT should be done and not on HOW to do it. It also makes it easy to do many of the common tasks which the business computer programmer wishes to do. HPBB has both an interpreter and a compiler. Because of this, the programmer can productively develop programs in the interpreter using its extensive editing and debugging features and then compile it to get high performance.

In addition, there are many programmers who know BASIC. With little or no additional training they can program in HP Business BASIC using most of its special features.

2.1 ADVANTAGES OF HAVING BOTH AN INTERPRETER AND A COMPILER

In non-interpretive languages, much of the information about a program is not present when the program is running. This causes problems when debugging either because debugging is non-symbolic or it is very slow (because the debugger may be another process which has access to the symbolic information). In either case it is not possible to change the program without another lengthy recompile. Since HPBB has an interpreter the development cycle can be shorter than when using "Compiled-only" languages such as COBOL and PASCAL. This is because the programmer edits his code and debugs it in one environment instead of editing his source, compiling it, PREPing it (also known as linking) and then finally running it with a debuger. Syntaxing is done only once, the first time it is entered, instead of during each and every compile. Syntax errors are displayed when the program is being editing. This allow the programmer to correct the problem right away instead of waiting for the compile to finish and then looking through a listing.

HPBB has numerous editing commands, many of which are found in dedicated text processors. The programmer can MOVE lines from one part of the program to another or make a COPY of one or more lines. One can FIND all occurrences of a character string within the program and may CHANGE those occurrences to a different character string. There are also commands for line editing (MODIFY and REDO) and a command to DELETE lines. As is traditional in BASIC implementations, a line can be replaced simply by entering the new line with the existing line number.

Adding many lines of text is simplified by the AUTO command, which provides ascending line number prompts. The RENumber commands will renumber all or part of a program. The INDENT command will "pretty-print" the program according to the programmer's specifications. The program can also be SCRATCHed, or completely deleted.

Programs can be saved as ASCII text or in a special file format called BSAVE. These files contain the program in the internal format used by the interpreter. It is this file format that is read by the compiler resulting in better performance of the compiler. There are several commands for managing the files of HPBB programs. The GET, LINK and MERGE statements which can be used to bring programs in from the disc, and the GET SUB and DEL SUB statements which bring in and delete subprograms.

When listing a program, the LIST TO command will send the listing to a file or an output device. There is also the LIST SUBS command which will list the subunits in the program. There are many ways of describing what portions of a program to be LISTed, MOVEd, CHANGEd. Some examples are ALL, FIRST, MAIN, LAST, line numbers, line labels, subunit names, + offset, - offset and ranges.

The SECURE command prevents a program from being LISTed or SAVEd as an ASCII file and the RUN-ONLY command prevents a BSAVE file from being edited or listed. As soon as a "RUN-ONLY" file is loaded, it automatically starts to RUN and as soon as it pauses or stops for whatever reason, the program is SCRATCHed from memory.

When an interpreted program is suspended during execution (either by the PAUSE statement, pressing an control-Y or by encountering an untrapped run-time error), variables and program lines may be displayed and modified. Execution can then be resumed with the CONTinue command. Variables can be TRACEd (and UNTRACEd) so that changes in their values are reported at run-time. To clearly show the execution flow of a program, lines can be also be TRACEd. The programmer may also execute most of the statements in HPBB (such as GOTO, GOSUB, CALL, ect.) as commands from the keyboard. Programs can be run in STEP mode which displays each line and executes lines one at a time. There is also the HOP command which allows you to single step a routine without having to single step all of the subroutines it calls. The INFO, FILES and CALLS commands give information on the current status of the interpretive environment such as what files are open and what are their current file pointers, what calls have been made and the current state of the OPTIONs. This is a vast improvement over the long wait encountered during program development when changing a program using non-interpretive languages.

There is an on-line HELP facility provides immediate information on syntax and functionality of all of the HPBB statements and functions. The HELP facility can also correct spelling mistakes so that it will find the right topic and/or subtopics even if they are slightly misspelled.

The interpreter is always in 'calculator mode', which allows the programmer to execute expressions entered from the keyboard. The HPBB compiler is directly accessible through commands in the HPBB interpreter. These commands are the COMPILE, COMPPREP and COMPGO commands. By typing COMPGO, HPBB will save your current program in a temporary file, compile it, prep it and then run it.

A string can be specified when running the interpreter and it will be executed when the interpreter first starts.

2.2 SPECIAL FEATURES OF THE LANGUAGE

HPBB has many features that allow the programmer to spend time on WHAT should be done and not on HOW to do it. These features make HPBB a very advanced system combining a language, an operating system and a language library, all in one.

In addition to some of the standard control structures such as GOTO, GOSUB, FOR-NEXT ON GOTO, GOTO OF, ON GOSUB, GOSUB OF and IF-THEN, there are statements and constructs to support structured programming, such as IF-THEN-ELSE, WHILE-ENDWHILE, REPEAT-UNTIL, SELECT-CASE, LOOP-ENDLOOP-EXIT LOOP and enhanced FOR-NEXT. This results in making the code more understandable than using IF THEN GOTO statements (and is faster also).

There are many statements to change for environment of HPBB. These include the OPTION DECIMAL, OPTION REAL, OPTION INIT, OPTION NO INIT, OPTION BASE 0, OPTION BASE 1, OPTION TRACE, OPTION NO TRACE, OPTION DECLARE, OPTION NO DECLARE, WARNINGS ON, WARNINGS OFF, CWARNINGS ON, CWARNINGS OFF, GRAD, RAD, DEG, DEFAULT ON, DEFAULT OFF, STANDARD, FLOAT and FIXED statements.

When dealing with the compiler there are the COPTION SEGMENT, COPTION NOLIST, COPTION LIST, COPTION LINES, COPTION PAGE, COPTION RANGE CHECKING, COPTION NO RANGE CHECKING, COPTION TITLE, COPTION WARN, COPTION NOWARN, COPTION ID TABLES, COPTION NO ID TABLES, COPTION LABEL TABLES, COPTION NO LABEL TABLES, COPTION MAXGOSUBS, COPTION PAGESUB, COPTION TITLESUB, COPTION SET ERRL, COPTION NO SET ERRL, COPTION REDIM, COPTION NOREDIM, COPTION ERROR HANDLING, COPTION NO ERROR HANDLING and COPTION MAXFILES statements.

It is possible to break up a program into separate subunits. There are three kinds of subunits: MAIN, subprograms and multiline functions. MAIN is the subunit that is executed when a program is run. Subprograms are subunits that are CALLed from other subunits. Multiline functions are subprograms that return a value. Both subprograms and multiline functions can have parameters. These parameters can be files or any of the data types supported in HPBB and can be scalars or arrays. Global data can be shared through COMMON. COMMON can be either named or unnamed. It is also possible to have single line functions within a subunit, although they use the subunits variables, can only contain expressions and have limitations on the parameters. Virtual program space allows large applications to be run when written in a modular fashion. Even though memory on the HP3000 may be limited, the HPBB interpreter can have a large program running in it and will only bring the current subunit into memory.

Run-time errors and interrupts from the keyboard (control Y, also known as HALT) can be easily trapped by the program and appropriate action taken. The statements that support this capability are ON HALT, ON ERROR, ON END #, ON DBERROR, OFF HALT, OFF ERROR, OFF END #, CAUSE ERROR and OFF DBERROR, as well as the ERRM$, ERRL, ERRN, ERRMSHORT$ functions. The ON and OFF statements arm and disarm the trap handling mechanism. If it is armed then when a certain trap occurs, the action specified in the ON statement would be executed. For example, if there is an ON ERROR CALL Error_sub, then when an error occurred a CALL to Error_sub would be executed. Only three type of action can occur in a ON statement: GOTO, GOSUB and CALL (without parameters). If an ON END # is active then when the file specified by the number following the # gets an END OF FILE error, the ON END action will be taken. If a ON DBERROR is active then any database statements without a STATUS clause which get a error will cause the ON DBERROR action to be taken.

HPBB contains statements which provide access to the IMAGE database management intrinsics. They are the DBCLOSE, DBDELETE, DBERROR, DBEXPLAIN, DBFIND, DBGET, DBINFO, DBLOCK, DBMEMO, DBOPEN, DBPUB, DBULOCK and DBUPDATE statements. These statements use keywords to simplify the IMAGE calls and to make IMAGE-based applications more readable and maintainabl. The programmer no longer needs to append semicolons and the end of strings, and then set the string length when returning from the intrinsic.

Any error encountered in the statement can be caught by HPBB's error handling facility if the STATUS clause is not specified. The ON DBERROR and OFF DBERROR error handling statement can be used to customize error handling for databases. There is the PACK, UNPACK and PACKFMT statements for packing and unpacking data into strings.

There is a formatter for performing formatted output which is very rich in its features. This is accessed by the PRINT USING, PRINT # USING, DISP USING and IMAGE statements.

HP Business BASIC provides the capability to call system intrinsics and programmer-supplied routines written in PASCAL and SPL. This is done by the EXTERNAL, GLOBAL EXTERNAL, INTRINSIC and GLOBAL INTRINSIC statements and the CCODE function. HPBB subunits can be compiled for performance reasons and called from interpreted routines. When calling HPBB routines, features such as COMMON and error-trapping are available. This provides a way to speed up an application or utility even though it is not completely compilable. There are two routines, BB_SORT_IT and BB_MERGE_IT, which allow the HPBB programmer to call the SORT/MERGE intrinsics.

When RUNing HPBB programs from the interpreter it is possible to specify an INFO string. This is the same as the INFO string on the MPE :RUN command for compiled HPBB programs and is accessible by the INFO$ function from both interpreted and compiled code.

Statements exist to issue MPE commands and to run other programs from within an HPBB program. These are the SYSTEM (or ":") and SYSTEMRUN (or ":RUN"). SYSTEM will execute a command on the HP3000 and SYSTEMRUN will run another program. You can specify all of the parameters of the MPE RUN command in addition to a NOSUSPend and a PRI= parameter. These tell HPBB not to suspend itself while the child process is running and at what process priority the child process is to run, respectably. Both of these statements can have STATUS clauses which indicate the result of the action done.

Long identifier names and alphanumeric labels improve program maintainability. Gone are the days of one or two identifier names. Variables, line labels, subunit names, etc. can have up to 63 characters.

There are numerious builtin functions in HPBB. Besides those mentioned elsewhere in this paper, there are functions for string processing, bit manipulation, trigonometric operations, accessing the system clock and other useful things. They are the CHAR$, WORD, DEBLANK$, COMPRESS$, TRIM$, POS, SCAN, LWC$, UPC$, RPT$, RTRIM$, LTRIM$, VAL$, VAL, SHIFT, ROTATE, BINAND, BINCMP, BINOR, BINXOR, BITLR, BITRL, SIN, COS, TAN, ACS, ASN, ATN, PI, DAT3000$, DAT$, TIME$, CLOCK, CPU, TIME, BRK, CEIL, CTL, FRACT, LGT, MAX, MIN, NUM, REAL, SHORT, INTEGER, SINTEGER, RND, SGN, SPA, SQR, SKIP, SREAL, DECIMAL, SDECIMAL, TAB, TASKID, USERID, ROUND, DROUND, EXP, REVISION, VERSION$ and ABS functions.

Seven data types provide flexibility and efficiency in data structure design. These types are long and short INTEGER, long and short floating point REAL (using binary representation), long and short floating point REAL (using binary coded DECIMAL representation) and strings. The names of thses types are INTEGER, SINTEGER or SHORT INTEGER, REAL, SREAL or SHORT, DECIMAL, SDECIMAL or SHORT DECIMAL, and strings. The binary representation of floating point REALs uses the machine instructions of the HP3000 resulting in high performance. The binary coded DECIMAL representation of floating point REALs provides higher accuracy because it can represent base ten numbers exactly (the binary representation occasionally has some small round off errors when converting from base 10 to base 2 and then back again) However, there is a performance penalty for using DECIMAL and SHORT DECIMAL. When an expression is evaluated which contains different type of numbers, they will be automatically converted to a matching type. The possible operators are plus, minus, multiplication, division, integer division (DIV), MAXimum, MINimum and remainder (MOD). There are also the relational operators <, >, <>, =, <= and >=. The logical operators <, >, <=, >=, <> and = can also be used in expressions. They return 1 for true and 0 for false. The boolean operators AND, OR, NOT and XOR can also be used. They consider 0 to be false and anything else to be true.

All of these types are available as scalars or as arrays. The maximum number of dimensions in an array is six. There are several statements and functions built into HPBB for dealing with arrays. Using these statements will result in more understandable and better performing code. The MATRIX statements are MAT READ, MAT PRINT, MAT INPUT, MAT READ #, MAT PRINT #, MAT INPUT #, MAT PRINT USING, MAT PRINT # USING and MAT assign. The MATRIX functions are CON, ZER, SUM, ROW, COL, IND, INV, TRN, CSUM, RSUM, DOT and DET.

For storing constants, there is the READ, MAT READ, DATA and RESTORE statements and the DATABUF function.

HPBB provides several statements and functions for accessing files on the system. These are the READ #, PRINT #, MAT READ #, MAT PRINT # and LINPUT #, CREATE, ASSIGN (open), ADVANCE, PURGE, RENAME, COPY, LOCK, UNLOCK, UPDATE, and POSITION statements, and the SIZE, SLEN, WRD, REC, TYP and FNUM builtins. There is a special HPBB file type available (BDATA) which allows data to be stored with type information to facilitate later retrieval. In addition, MPE's ASCII and BINARY file formats are fully supported. For BDATA files, the I/O can be directed to a specific word within a record. It is also possible to have all reads and writes be encrypted. The FILES ARE IN statement provides a way to tell HPBB to get all files from a certain group and account.

It is possible to redirect output from the terminal to other files. The SEND OUTPUT TO statements will redirect the result of normal PRINT statements to files or devices. The SEND SYSTEM OUTPUT TO statements will redirect the result of normal HPBB commands such as LIST and TRACE to files or devices. The COPY ALL OUTPUT TO statements will copy all of the output of HPBB to files or devices. Some of the files or devices that can be specified are "filename", NULL, DISPLAY and PRINTER. PRINTER is defined in the configuration utility mentioned below.

The COPYFILE command allows you to copy one file to another. If no "to file" is given, the file will be copied to the terminal.

When doing input and output to the terminal there are many features that can be used. These include the PRINT, INPUT, TIMPUT, PRINT USING, DISP, DISP USING, LINPUT and BEEP statements and the BUFTYP function.

When printing or inputing, whether it is from a file, the terminal or the DATA statements, it is possible to have a simple FOR-NEXT loop in the I/O statement. This is know as the "embedded FOR loop".

There is the CAT or CATALOG statement which allows you to list the names of the files in the system. It is possible to specify the TYPE of file to be searched for as well as having wild-cards in the name.

There is a configuration utility for customizing the default OPTION settings to the programmers needs.

Conversion tools are provided for BASIC/3000 and BASIC/250 applications. These will be discussed in a later section.

HPBB is localizable to the local languages and customs of many countries around the world.

3.0 PERFORMANCE TUNING

In order to make the best use of any tool it is necessary to understand how it works and how to make the best use of its features. HP Business BASIC improves the productivity of the programmers using it by doing much of the work for them. The penalty for this is that performance may suffer because HPBB may be doing work for features that are not used. Here is a list of things to do to help improve the performance of HP Business BASIC applications and utilities. While compiled HPBB offers good performance and in some cases may be faster than other languages, it is normally somewhat slower that other languages like PASCAL, SPL and FORTRAN for CPU-bound programs. This is because of the additional code generated by the compiler to support the special features of the language. When a program is I/O or IMAGE bound, the difference may be insignificant.

3.1 COMPILING

Compile all or part of your program. This may be the easiest thing to do and performance improvements for most applications will be very substantial. Certain I/O-bound applications, however, will show little improvement. The COMMAND, LINK, GET, GET SUB, DELETE, DEL, MERGE, RE-SAVE, PAUSE, SAVE, SCRATCH, SECURE, STORE and TRACE statements cannot be compiled. If you cannot compile all of your application the compile as many subroutines as possible and call them from your interpreted program. It is therefore possible to eliminate these statements from low-level routines and compile those, resulting in faster execution of the interpreted main program. Compiled programs also result in a much lower demand on operating system resources (memory, disc, etc.). Please be careful of any of the small differences in behavior between compiled code and interpreted code, especially running the two together.

There are a number of compiler options that affect the size and therefore the execution speed of the compiled program. They are COPTION NO RANGE CHECKING, COPTION NO REDIM, COPTION NO SET ERRL and COPTION NO ERROR HANDLING. COPTION NO RANGE CHECKING causes the compiler to not emit code that will produce a run-time error if a number or an array index is out of bounds. COPTION NO REDIM cause the compiler to emit code which assumes that array bound will not be changed at run-time throught the use of the REDIM statement or the redim option on some of the MAT statements. Any attempts at redimensioning are illegal. COPTION NO SET ERRL causes the compiler to not emit code to update the value returned by the ERRL function. COPTION NO ERROR HANDLING causes the compiler not to emit code after each statement to check whether errors or HALTs have occurred and to process the appropriate ON HALT, ON ERROR, ON DBERROR statement, the use of which will be illegal. COPTION NO ERROR HANDLING also causes COPTION NO SET ERRL. If you use these COPTIONs, then use the STATUS clauses of the statements in your program.

Try to segment your program using the COPTION SEGMENT statement so that there are less intersegment calls. Large segments take longer to load than small segments.

If, after you have compiled you program and tried some of the suggestions in the next section, you still need better performance there is one thing left to do. There are some things which may be more efficient if written in another language such as PASCAL.

3.2 USE THE HIGH PERFORMANCE FEATURES

If, when writing an application or utility you pay attention to what features you use, then you will be able to increase the speed of your programs.

The type of numeric data you use can have a major impact on the performance of your program. If possible have all numbers in a expression be the same type. Conversion between numeric types can take longer than the actual numeric operations. Constants are stored as DECIMAL, REAL, INTEGER and SHORT INTEGER. As a result, any other type will require conversion when combined in expression with constants. DECIMAL, SHORT DECIMAL and OPTION DECIMAL use HPBB's uses binary coded decimal instead of binary for its internal representation. There is limited hardware support for this and so these operations are slower. If at all possible use another data type. All SHORT DECIMALs are converted to DECIMALs whenever they are used, and as such, should only be used if you are short on space. When going between DECIMAL and REAL, all conversions go through ASCII, and if going to/from SHORT REAL there is an additional conversion. So, going from SHORT DECIMAL to SHORT REAL causes three conversions.

Don't have a production program which has self modifying code. Self modifying code is code that changes itself through the use of statements like DELETE, GET, GET SUB, RE-SAVE, SAVE, SCRATCH, LINK and MERGE. Also the COMMAND statement creates code dynamically. These statements are slow if the accessed code is in BSAVE format and VERY SLOW when using ASCII files. It is alright to uses these in development, but they will cause the interpreter to be very slow and will not be compilable. The most efficient size of a subprogram is about 200 to 400 lines long.

HP Business BASIC may do a lot of work for the programmer when executing some of its features. This work may be to support a subfeature that you are not using. If this the case, the programmer is may be better off rewriting the code. An example might be writing strings to an ASCII file. The programmer might want to call intrinsics directly, e.g. calling FOPEN, FREAD, FWRITE, FCLOSE directly instead of using ASSIGN, READ # and PRINT #. Another example might be to call the CREATEPROCESS intrinsic to run another program, if there are only simple parameters. Please note that it is not always easy to call intrinsics.

There are several features in the language which do a large common operation. They were implemented as efficiently as possible and will probably be faster than if the programmer implemented the action. Some examples of this are the COPYFILE, MATRIX statements and the embedded FOR-NEXT loop in I/O statements. Also try to use the structured and IMAGE statements. They make the code easier to understand and are faster also.

When choosing a file type to use be aware that fixed record size file are faster the variable record size files, but they use up more space. When saving a HPBB program try to use the BSAVE file format. It is faster and smaller than ASCII or BDATA, and is required by the compiler.

GOSUB is faster than CALL. Long parameters lists take longer. Make loops contain as few statements as possible.

Only specify the portions of COMMON which are required by the subunits. Also make each line do as much as it can. Even though this may make the program slightly less understandable, it eliminates the overhead associated with processing a line.

Whenever possible try to evaluate expressions outside of the program. By using the calculator mode and only putting the result in as a constant, the program will run faster.

Removing comments from the production version of a program which is interpreted will make it run slightly faster. This has no effect on compiled programs.

4.0 RESOURCE REQUIREMENTS

The interpreter uses a lot of code and extra data segments. This will put a load on memory management system. Adding memory, and/or compiling you applications will fix this.

Some of the features in the interpreter can take up a lot of memory, resulting is less user memory available than was on BASIC/3000. This is not a problem for compiled code.

HPBB does use the PASCAL HEAP. Be careful using MARK and RELEASE.

5.0 PORTABILITY

HP Business BASIC was written to be as compatible as possible with BASIC/250 and BASIC/3000. As a result HPBB includes tools to assist in the conversion of BASIC/250 and BASIC/3000 applications.

The conversion of BASIC/250 applications involves the transfer of files from the HP250 to the HP3000 (using data communications tools included with the BASIC/250 conversion package) and conversion of those files on the HP3000. The conversion package will allow the transfer and conversion of BASIC/250 source programs, data files, databases and forms files. The run-time behavior of each HPBB statement has been carefully designed to duplicate, as closely as possible, the behavior of the corresponding BASIC/250 statement. However, there are differences and omissions of some BASIC/250 features due to machine dependencies. In addition, there are a number of 250 subsystem statements (in particular those statements in the FORMS/250, Report Writer/250 and SORT/250 subsystems) which are not currently included in HPBB. Therefore, after an application has been run through the conversion utility, manual conversion will be required in most cases before the application can run successfully with HPBB.

Since HPBB is highly compatible with BASIC/3000 on a statement-by-statement basis, the conversion is accomplished mainly through use of the conversion utility which can be included, as an option, with the purchase of HPBB. Difference in the organization of programs may result in the necessity for some manual modifications in order to make those applications run with HPBB. The conversion utility will convert BASIC/3000 programs and data (BASD) files.

HP Business BASIC will be the standard for BASIC on all future HP commercial system.

6.0 CONCLUSION

In this paper we have seen how HP Business BASIC can add to the productivity of programmers using it. This is because of the many features and the interpretive environment it provides. We have also seen how to use these features in a cost effective manner. There is a myth that BASIC is just a toy language and not to be used for real work. Already that notion has been proved false on the HP250. Hopefully it will be proved false again for HP Business BASIC/3000.

7.0 REFERENCES

For more information on any of the features in HPBB please refer to the HPBB Manual series. This is a set of five manuals which constitute the user documentation of HP Business. There is also an on-line HELP facility in the interpreter which provides information on syntax and functionality of all of the HPBB keywords and statements.

HP Business BASIC Programmer's Guide Part No. 32115-90007

This guide is for those who want to learn how to program in HPBB.

HP Business BASIC Reference Manual Part No. 32115-90006

This manual describes all of the features of HPBB. It is for those who wish look up how an HPBB feature works.

HP Business BASIC Quick Reference Guide Part No. 32115-90008

This is a "Quick Reference" version of the reference manual.

BASIC/250 to HP Business BASIC Conversion Guide Part No. 32115-90010

This guide describes how to convert BASIC/250 applications into HPBB applications. It also describes the incompatibilities and the conversions which take place.

BASIC/3000 to HP Business BASIC Conversion Guide Part No. 32115-90009

This guide describes how to convert BASIC/3000 applications into HPBB applications. It also describes the incompatibilities between the two BASICs.

3068. MPE Disc Caching

Bryan Carroll
Marketing Engineer
Computer Systems Division
Hewlett Packard
19447 Pruneridge Avenue
Cupertino, Ca.95014

MPE Disc Caching
Introduction

Now that MPE Disc Caching has finally arrived, it has become the job of many users to determine what is happening inside this box we call Disc Caching and learn how to take full advantage of its capabilities. Disc Caching is being widely used by HP3000 users. There is little user documentation available for internal flow and organization to help understand how Disc Caching works. Frequently asked questions by users attempting to understand Disc Caching include: What is a Random/Sequential Fetch Quantum?, What is a Serial Write Queue? or What happens when you issue a STARTCACHE/STOPCACHE command?.

These are excellent questions that can help a System Manager tune his/her system, a programmer increase the integrity of an application system, and an operator understand what it means to start or stop Caching on one or more discs. Based on my experience with the MPE Operating System, I would like to share what I have learned and am continuing to learn about the Disc Caching Subsystem. This paper will focus primarily on the implementation details of Disc Caching including some guidelines for its successful use.

History

During the development of the MPE IV Kernel, algorithms were developed that effectively managed the resources and optimized performance for the existing HP3000 family. Since that time, the Series 64 was introduced with a processor twice as fast and main memory size four times that of the previous top of the line HP3000. The Series 64 performance was found to be very sensitive to disc access times and relatively insensitive to main memory size. The performance bottleneck of the Series 64 was then found to be the disc subsystem. Efforts were then focused toward increasing the disc throughput. Figure 1 illustrates a typical storage hierarchy with the cost per byte and access speed increasing as you move up the scale and capacity increasing as you move down the scale. There have been four different traditional approaches for addressing the bottlenecks between storage levels. These four approaches are: increase the management algorithms of the levels, increase the capacity of the faster level, speed up the access time of the next lower level, or introduce a new level into the system. The HP3000 research and development lab went to work with these four ideas to determine which, if any, of these we could implement on the HP3000 family.

Standard Computer System Storage Hierarchy

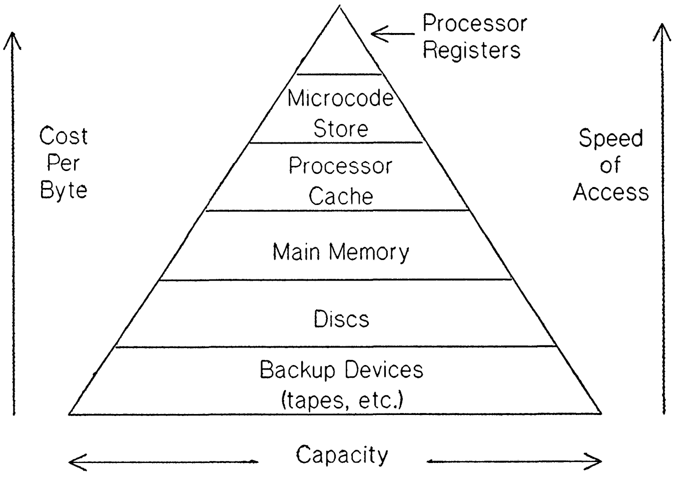


Figure 1

The next lower level of the hierarchy was the disc subsystem and one of the tasks was to determine if we could speed up the access times of the disc subsystem. Processor speeds have been increasing and the cost per byte of semiconductor memory has been dropping by orders of magnitude but the disc technology has not

been keeping up with the increases in these other areas. The gap between a semiconductor memory reference and a disc reference is currently about five orders of magnitude and growing. Although discs have been getting more and more dense, the access times have remained relatively constant with the current moving head technology. When a new technology is developed or when major advances can be made in moving head technology the gap between main memory and discs may be narrowed, but until that time, we will have to look elsewhere for improved disc throughput.

Another traditional solution to removing a bottleneck is to introduce a new level into the hierarchy. This level would have to be introduced into the gap between the disc and main memory to help the Series 64 bottleneck. Research was then done to investigate the possibility of a disc resident cache independent of any higher levels in the hierarchy. Introducing a new level at this point using either bubble or semiconductor memory technology has shown little cost effectiveness. Bubble memory has not been able to keep pace with the density improvements of semiconductor memory which has kept the cost per byte relatively high. Semiconductor RAM memory has become more and more dense and the cost per byte has been dropping but the cost per megabyte is still two orders of magnitude greater than that of discs. Until major technology changes occur, we can expect magnetic discs and semiconductor main memories to be dominant in computer storage hierarchy.

The remaining two approaches have been combined to give us disc caching. The main memory capacity on the HP3000 family has been increased to eight megabytes on the Series 64 and the Series 44 capacity has been doubled to four megabytes and it is likely that future HP3000s will have even larger main memories. The MPE IV memory manager has been enhanced with the addition of the Disc Caching module to allow what we know as disc caching.

Overview

Disc caching on the HP3000 family is an optional feature that takes advantage of excess CPU power and excess main memory to keep frequently referenced portions of disc buffers in main memory. The Disc Caching Subsystem monitors memory usage and uses as much free memory as it can to increase the chance that the next requested disc record will already be in main memory. There is no permanently allocated portion of memory set aside for disc caching. The algorithm monitors the current amount of memory required for user and system data and code segments and adjusts the amount of memory it uses for Disc Caching accordingly.

Disc Caching takes advantage of its knowledge of the file system to maximize the chance that the next requested record will already be in memory. The goal of the Disc Caching Subsystem is to maximize the number of times a request is made to read a record which is in memory (read hit) and to minimize the number of times that a request is made to write a record to disc which already exists in main memory and is currently being written out to disc (write hit). In both of the above cases, a disc access will be required to satisfy the request. In the HP3000 family, a main memory move instruction, which will be executed when a disc record is found to already be present in memory when a process requests it, takes from 3 to 5 milliseconds. A disc access, which will be required when a requested record is not already in main memory, takes from 30 to 40 milliseconds, which by comparison is a very long time.

Disc Caching can be enabled or disabled on a disc by disc basis. When Disc Caching is enabled for the first disc, all caching related resources are created and likewise when Disc Caching is disabled on the last disc, all related resources are deleted from the system. This concept will ensure that Disc Caching will not introduce any overhead when it is not enabled.

Disc Caching Data Structures

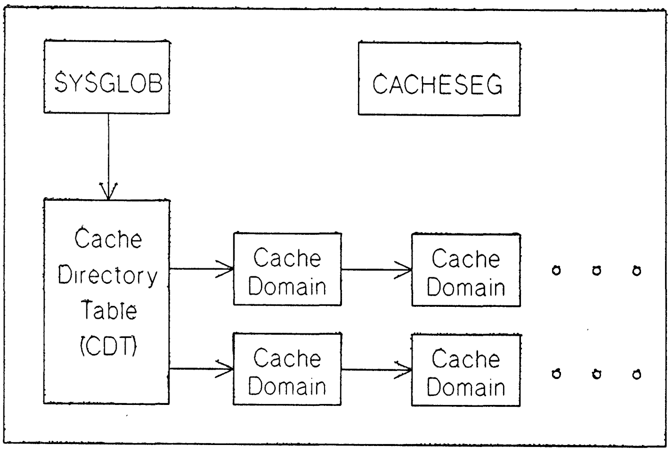


Figure 2

When caching is enabled on the first disc, a Cache Directory Table (CDT) is created to keep track of all caching related data. The CDT maintains an entry for each disc that has been enabled for caching. The CDT also maintains pointers to the portions of disc that have been saved in main memory (cached) in hopes of satisfying a disc request without having to access the disc. The portions of disc that are memory resident are called cache domains and are variable in size depending on the file structure and the available memory. The cache domains are also connected by a linked list for each disc. This list is linked in increasing disc address order.

Data Structures

As we just mentioned, the Cache Directory Table (CDT) keeps track of all caching related information. There is one CDT for each system with caching enabled. The CDT is pointed to by three fixed low memory locations. These locations are in the area of memory known as System Global (SYSGLOB) and includes a one word Data Segment Table (DST) number, a one word bank address, and a one word bank offset address for the CDT. These words will be zero for a system without caching or when caching is not enabled.

The CDT contains three different types of entries: the Header, Device, and Mapped entries. There is one Header entry in each CDT that contains some global CDT statistics and pointers. The information found in a Header entry includes the number of entries in the table, the first free entry, the number of logical devices currently cached, etc.

There is one Device entry for each logical device currently being cached. The device entries contain information such as the logical device of the disc being cached, pointers to the cache domains, caching statistics, etc. A Device entry is created for every successful STARTCACHE command and deleted for every successful STOPCACHE command. The SHOWCACHE command uses the information found in the Device entries to report caching statistics. Each Mapped entry corresponds to a cache domain. The mapped entries are searched by the read and write logic to determine if there is an I/O already in process for the desired disc location. The mapped entries are for every physical any logical I/O performed by the Disc Caching Subsystem. The mapped entries hold information such as the sector start and stop disc address of the I/O, the target memory address of the cache domain, and a number of flags. Each mapped entry pertains to a device entry and all mapped entries belonging to the same device entry are linked together.

A cache domain is a main memory resident temporary storage area for data brought in from the disc. A cache domain may reside in any available area in memory and is very similar to an extra data segment but does not require a data segment table (DST) number.

We have briefly covered all the Disc Caching specific data structures. We will now walk through the Disc Caching commands and see how the resources are used by the Disc Caching subsystem.

STARTCACHE

The STARTCACHE command is responsible for all operations necessary to enable Disc Caching on the specified disc. The STARTCACHE command executor can be broken down into three different sections. First, some checks are made of the state of caching; second the resources are obtained; and third, the data structures are completed with valid data and caching is allowed to begin.

There are several checks that must be made before any resources can be allocated. The first check is made to see if caching exists on this system and if the specified device will support caching. The device specified in the STARTCACHE command must be a disc. At this writing, all random access discs will support Disc Caching. Once we have determined that the system and logical device will support caching, we determine if caching is currently being used on the system or if we are the first command to enable caching. If caching is not currently active, we must build the CDT and lock it into memory. The CDT cannot be swapped out to virtual memory since this would defeat its primary purpose of reducing disc transfers. In addition to building a CDT, the caching code segment (CACHESEG) must also be loaded into memory and locked for the same reason. The fixed low memory locations are then updated to point to the CDT memory location for later use by the Disc Caching subsystem. If we are successful at all of these checks, we then proceed to collect the needed resources.

Some of the more general resources have been established in the checking phase. We must now gather specific data entries to allow caching to be enabled. The primary entry we are interested in is a device entry in the CDT table. There is no upper limit on the number of device entries that can exist in the CDT. This number is only limited by the maximum number of supported disc drives which is currently 24 per system. If for some reason, we are unable to obtain a device entry in the CDT, a system failure 1009 will result. A device entry is obtained and initialized to zeros, and we now begin the business of completing the entry with valid and useful data.

The device entry is completed and linked in with any other device entries that may exist at this time. All counters are initialized to zeros to ensure integrity of the counters, and the pointers to the mapped entries and cache domains are initialized to zeros since these entries do not exist for this device yet. Some control words and pointers in the header entry are updated to reflect the addition of a new device entry. The final value set which makes caching available on the specified device is a bit in the Device Information Table (DIT) for this device. The bit is set on to indicate caching is enabled and the next disc request to this disc will use caching code to resolve it.

Cached Read

The next disc request that comes down from the file system will be intercepted by the caching routines. If this is a user I/O, the operation will be performed on the users stack and will require approximately 255 (%377) words of stack space to execute. The first caching routine to execute is called CDT'ATTACHIO and will perform checks of the many parameters that are passed to it as well as some more checks specific to disc caching. If the tests pass, we will check to see if caching is in the process of terminating (STOPCACHE command just executed). If this is the case, we will not perform the I/O but instead will wait until the STOPCACHE command has finished executing and will return to perform a physical I/O. If all is well, we will continue by building an entry into a table called the Logical Disc Request (LDR) table. The LDR is a specially formatted Disc Request Queue (DRQ) entry and is not part of a separately configurable table. The LDR changes the DRQ entry format for the specific needs of the Disc Caching subsystem. Once this entry is built, a separate routine called REQUEST'CACHE is called to determine if the requested data is already in memory.

REQUEST'CACHE will attempt to satisfy the request. If it is successful, we have saved a physical access and will return control to the current process without requiring the process to block. If REQUEST'CACHE is not successful, a physical disc access must be initiated to satisfy the request and the process must block while this is done. REQUEST'CACHE will begin looking for the requested data with the device entry in the CDT. The device entry contains a pointer to the first and last mapped entry for this device. Each mapped entry will be searched sequentially for a match with the requested address and length. If a match is not found in the mapped entries, the cache domains are then searched sequentially. HP has investigated implementing other methods of scanning the mapped entries for a specific address in an attempt to increase the throughput of the system, but all the methods investigated so far proved not to provide substantial throughput increases.

As a result of the search, one of four things could happen; the requested data could be found in a cache domain (hit), the data could be found in a mapped entry, the data could not be found (miss) or part of the data could be found (partial hit). If all of the requested data is found, it is moved to the user's data area and control is returned back to the current process who continues executing without being rescheduled. If the data was found in a mapped entry, it means that an I/O is already in progress for this data. If necessary, the disc I/O priority is raised to the priority of the requesting process and the process blocks awaiting the I/O completion. If the data could not be found, or if only part of the data could be found, the request cannot be satisfied without a physical disc access and the process must block. In this case, a new block of memory is obtained, or if memory pressure exists, the least frequently accessed memory region is flushed to disc, if necessary, and the region is allocated. Once the new cache domain is allocated, the physical disc I/O is performed to read the data from the disc into the new cache domain. The physical disc I/O will usually move a data block larger than the requested data. Data that is close to the requested data is also read into the cache domain to satisfy any requests for that data that might be made.

After the data has been transferred to the user, some counters are updated in the device entry to track how successful the disc caching subsystem has been. These counters are used to compute the numbers displayed by the SHOWCACHE command to be discussed later.

Cached Write

A cached write is very similar to a cached read in its operation but the goal of the writing strategy is the opposite. The goal of the read logic is to maximize the number of times that the requested data is already in memory. The goal of the write logic is to maximize the number of times that a record being sent to the disc is not found in memory. The goal of both systems is to minimize the number of physical disc accesses which require a process to block. When the caching subsystem receives a write request, the mapped entries in the CDT and the cache domains are searched just as they are for a read request. This search also yields the same four possible results; the requested data was found in a cache domain (hit), the data was found in a mapped entry, part of the data was found in a cache domain (partial hit), and the data was not found in any cache domains (miss). If the data was not found in any cache domains, a new cache domain is acquired as with the read miss situation and the data to be written to the disc is put in this new cache domain. The new cache domain is linked into the linked list of cache domains for this logical device and the process is allowed to continue without blocking. A physical write must still be scheduled for this cache domain but this write can take place at a background priority. A background priority means that the physical I/O will not compete for disc accesses with processes that are blocked waiting for a physical I/O to complete. This has the effect of increasing overall throughput because the process issuing the write does not have to wait for this data to be posted to the disc to continue. The data will remain in the cache domain until a time that no higher priority physical disc request is pending for this disc. It is to our advantage then, to minimize our write hits as well as to maximize our read hits.

In the case of a partial hit, a total hit or a hit in a mapped entry when a write function has been requested, more work must be done. In all cases, the current process must block for a physical write to complete before continuing. The cache domain that caused the hit during the search will be flushed to the disc and then the same logic used for a write miss described above will be used to add the data being written to a cache domain. When flushing the cache domain to the disc, a Disc Request Queue (DRQ) entry must be obtained and completed and passed to the disc driver. If this cache domain has been previously written to, a DRQ entry may already exist at a background priority so a search must be made of the DRQ. If an entry is found for the cache domain we need to flush to the disc, its priority is raised to the priority of the process forcing the disc I/O and the process is blocked pending the I/O completion. When the I/O completes, the cache domain that was just flushed is filled with the data that the blocked process is writing and a background disc write is initiated for the new cache domain. Once the background write is initiated, the process that initially requested the write is unblocked and allowed to continue processing. The blocked process does not wait for the data it is sending to be posted to the disc.

CACHECONTROL

We have seen how Disc Caching gets started and how a typical read and write operation take place, however, you probably have many questions in your mind about how you can control Disc Caching. The CACHECONTROL (appropriately named) command and the FSETMODE intrinsic will give you some control over how things happen in the Disc Caching subsystem. The CACHECONTROL command has three options, SEQUENTIAL, RANDOM and BLOCKONWRITE.

The SEQUENTIAL and RANDOM options allow you to set a target number of sectors that you want Disc Caching to bring into a cache domain in memory every time it encounters a read miss. These are both called "fetch quantums". The size of the cache domain built, following a write miss, is always the size of the requested transfer so write requests are not affected by any fetch quantums. In the case of a read miss, Disc Caching will take the requested size and round it up. The value selected will be the largest even multiple of the requested size that is less than or equal to the fetch quantum, but never less than the requested size. This means that the requested size will always be brought into memory and usually several times more than the requested size will be brought in. The values of the fetch quantums can range from a low of only 1 sector to a maximum of 96 sectors. The default values are 16 sectors for the RANDOM option and 96 for the SEQUENTIAL option. Disc caching uses the file system information concerning the type of access the file is being used for to know which option to use on any given read. If the read request is a FREAD, Disc Caching will choose the SEQUENTIAL option value. If the read request is any other type of read, including FREADBACKWARD, Disc Caching uses the RANDOM fetch quantum.

The setting of these two values can vary your performance and total disc throughput, and there are many factors to consider in setting these values for your shop. We will address a few of the factors here briefly. In most cases, the default settings for both options will give the best results. These values were chosen after extensive testing using varying system loads and configurations.

The SEQUENTIAL option should usually be set as high as it can. If a file is being accessed sequentially, records will be read in sequence so the more records you can put into memory, the more disc accesses can be saved. One important note about sequential file access is that after the last record in the cache domain of a sequential file is transferred to the users stack, the cache domain is marked as a Recoverable Overlay Candidate (ROC) which means that the memory space used by that cache domain can now be used by some other operation if it is needed. This is the method the memory manager uses to determine who has been least recently accessed.

The RANDOM option gets a little more complex. There are several reasons why a default value of 16 was chosen, the most important of which deals with the internal disc resident hardware buffer found in the CS80 discs. There is a 4K byte hardware buffer built into the CS80 discs to reduce the time encountered because of latency (disc rotational delay) for small transfers. Any transfer requests greater than about 16 sectors will not use this hardware buffer and will cause the transfer to be slower. In some cases, this slightly slower transfer may be advantageous because of a possible higher hit rate, but you should consider the options and experiment with your system. The number of process stops displayed by the SHOWCACHE command is another indicator of system performance and in particular the Disc Caching performance. This number is often affected by the RANDOM fetch quantum value. If the process stops, as a percent of cache requests (process stops/cache requests), is around 13% or greater, this is an indicator of excessive process stops that may be due to an improperly set RANDOM fetch quantum. If the RANDOM fetch quantum is set too high, then you may be encountering many write hits which causes a disc access and a process stop. You may want to experiment with a lower RANDOM fetch quantum in this case to reduce the chance of a write hit.

The size of both fetch quantums also impacts the amount of memory that will be used for caching. If you are short on memory, it would probably be best to experiment with smaller fetch quantums to relieve any memory pressure that may be present. Since disc caching may be enabled and disabled on a disc by disc basis, you should look at the performance of each disc individually. In some cases it may be best to stop caching on one or two discs with a poor read percentage, read hit or high write hit rate, and release the memory used by those discs to increase the performance of the good performing discs even more. The bottom line concerning the fetch quantums is to experiment in your own shop with your own application mix running to see what settings give you the best performance.

The BLOCKONWRITE option of the CACHECONTROL command is an option that will allow you to sacrifice performance for data integrity. The BLOCKONWRITE option, if on, will tell the disc caching subsystem to always cause a write operation to block the requesting process until the data is recorded on the disc. This option will prevent the situation where data could be written by a program and held in a cache domain when a system failure occurs. This would result in the data being lost since memory contents are not recovered after a system failure. This option will always decrease performance (up to 30%) since unlike the case of a write miss, every write operation will cause a physical disc access to occur while the requesting process waits. This can be a sensitive issue which must be addressed by each individual shop.

The BLOCKONWRITE option on the CACHECONTROL command will establish the BLOCKONWRITE option for the entire system. You may also establish the BLOCKONWRITE option on a file by file basis with the FSETMODE intrinsic. After a file is FOPENed, a call to FSETMODE with the appropriate parameters will allow you to establish the BLOCKONWRITE feature even if the rest of the system is not using BLOCKONWRITE. Image with logging and KSAM both use the FSETMODE intrinsic to enable BLOCKONWRITE so you do not have to worry about those two subsystems.

Another option available with the FSETMODE intrinsic and disc caching is the Serial Write Queue (SWQ). Since a write operation does not always transfer the data directly to the disc, several write operations could all be waiting in cache domains at the same time. Since all of these requests will be queued waiting to be transferred to the disc at the same priority, they may not arrive at the disc in the same order that they were written in. In most cases this will not be important but in a few other cases it can be critical. It is for this reason that the Serial Write Queue was developed and made available. When the Serial Write Queue is specified for a file with the FSETMODE intrinsic, you are guarantee that all writes performed against that file will be written to the disc in the order they were issued by the program. The SWQ is a First In First Out (FIFO) linked list of entries in the Disc Request Queue. There is only one SWQ for the entire system so as an extreme example if the entire system used the SWQ, only one disc write would be serviced at a time even though there may be multiple discs, controllers, GICs, and even multiple Inter Module Buses (IMB's). This would cause extreme system degradation, so caution should be used in determining if the SWQ is necessary for your application.

Here are a few closing notes about BLOCKONWRITE and the Serial Write Queue.

1. Setting BLOCKONWRITE on globally with the CACHECONTROL command will disable the Serial Write Queue since all writes are occurring as they are requested in the order that they are requested.

2. Both Image and KSAM use BLOCKONWRITE and the Serial Write Queue for "important" transaction such as Intrinsic Level Recovery (ILR) for Image.

3. With BLOCKONWRITE off, the disc writes will occur after the requesting process has issued the write and continued its processing. If a write error occurs when the cache domain is finally flushed to the disc (i.e. a bad track), a system failure 650 or system failure 651 will result. If the same disc error occurred with BLOCKONWRITE on, or without caching enabled, a write error would be returned to the file system.

SHOWCACHE

The SHOWCACHE command is the only way to get Disc Caching statistics from the Disc Caching subsystem without purchasing any performance monitoring programs or performance consulting. The output from the SHOWCACHE command is, however, a very simple and accurate picture of the current status of the Disc Caching system. Figure 3 shows a sample SHOWCACHE output. The output shows one line per disc with a total line showing a summary of the combined Disc Caching subsystem.

```
DISC  CACHE    READ WRITE        PROCESS             % OF   CACHE
LDEV  REQUESTS HIT% HIT%  READ%  STOPS    K-BYTES    MEMORY DOMAINS
-----------------------------------------------------------------

1     264738   86    79    80    34625    575          7    349
2     113893   67    66    68    27575    337          4    132
11    165391   83    70    79    23714    715          8    308
12    715884   86    84    86    115000   1970         24   673

Total 1259906  84    78    82    200914   3597         44   1462

69% of user I/Os eliminated.
Data overhead is 374K bytes.
Sequential fetch quantum is 96 sectors.
Random fetch quantum is 16 sectors.
Block on Write = NO.
```

SHOWCACHE Output Figure 3

DISC LDEV is the logical device of the cached disc for which the following data applies.

CACHE REQUESTS are the number of times a process has requested a read or write operation to this disc, since Disc Caching was started on this disc.

READ HIT% is the number of times a read request was satisfied without a physical disc access divided by the total number of read requests made to this disc since caching was started on this device. You should try to maximize this number (i.e. greater than 60%).

WRITE HIT% is the number of times a write request required a process to block in order to satisfy the request divided by the total number of write requests made to this disc since caching was started on this device. You should try to minimize this number (i.e. less than 50%).

READ% is the total number of read requests divided by the total number of cache requests since caching was enabled on this device. The difference between this number and 100% is the write percentage. Caching will show the greatest performance gain when the read percentage is highest. PROCESS STOPS is the total number of times that a process had to be blocked because a physical disc access had to be performed. If the process stops, as a percentage of cache requests, reaches about 13% or greater, adjusting the fetch quantums may increase throughput.

K-BYTES is the number of bytes in thousands used by caching this disc. This is for your information only.

% OF MEMORY is the number of K-BYTES used by caching this disc as a percentage of the total memory available on the system.

CACHE DOMAINS are the total number of memory regions set up and being used to cache this disc. The average size of a cache domain for this disc can be calculated by dividing the number of K-BYTES by the number of cache domains.

The information at the bottom of the display indicates the status of the fetch quantums and the BLOCKONWRITE flag. In addition, the data overhead and percent of user I/O eliminated is calculated and presented here. The data overhead number includes the memory used by the CDT, the header and the trailer information for each cache domain. The percent of user I/O's eliminated is a calculation of the overall success of disc caching on this system. It is calculated by multiplying the overall read hit percentage by the read percentage and does not take into consideration any write operations transferred at a background priority.

When you are trying to determine what combination of Disc Caching settings work best in your shop, it is useful to try a setting for an hour or more and check your success with the SHOWCACHE command. Since the totals displayed in the SHOWCACHE command are not reset, you must stop and then restart caching before each test to initialize the counters. This can be a high overhead operation. A contributed program has been written and contributed to the INTEREX Contributed Library to help this situation. This program will lock the Caching SIR and zero all totals for all currently cached discs. This program should be used with care as you would any other contributed library program. The program is called ZEROTOT.

STOPCACHE

The STOPCACHE command is just like the STARTCACHE command only in reverse. The executor of the STOPCACHE command can be broken down into the same three sections that the STARTCACHE command was, except in the STOPCACHE command, they are executed in a different order. The first section performs some checks of the state of caching, the second section flushed all cache domains related to this device to the disc, and the final section releases all resources.

The first check is made to determine that caching exists on the system and that caching is enabled. Additionally, several checks are made against the system, such as verifying that the logical device number passed to the routine is a disc device and is currently cached.

When it seems that we can perform the requested function of stopping caching on the specified logical device, we post all "dirty" cache domains to the disc. A "dirty" cache domain is one in which the data in the cache domain has been changed after it was brought into memory so that the copy in memory is different than the copy on the disc. Next, the bit in the device information table (DIT) which indicates caching is enabled, is turned off for the specified device. This will prevent any more cache domains from being built until we are done with the STOPCACHE command.

When all the data has been posted to the disc and we have prevented any more data from being put into cache domains, we release the resources used by caching this device. This includes the device and all associated mapped entries in the CDT and all the cache domains. If this was the only disc currently being cached, we will also unlock and release the CDT, and unlock the caching code segment so that it can be unloaded as soon as we complete the STOPCACHE command.

Should I purchase Disc Caching

When we defined Disc Caching for the HP3000 family of computers, we said it takes advantage of excess CPU power and excess main memory. It is very important that you take a careful look at your system before attempting to use Disc Caching. In most cases Disc Caching will improve your system performance but in some cases, it could cause a performance degradation.

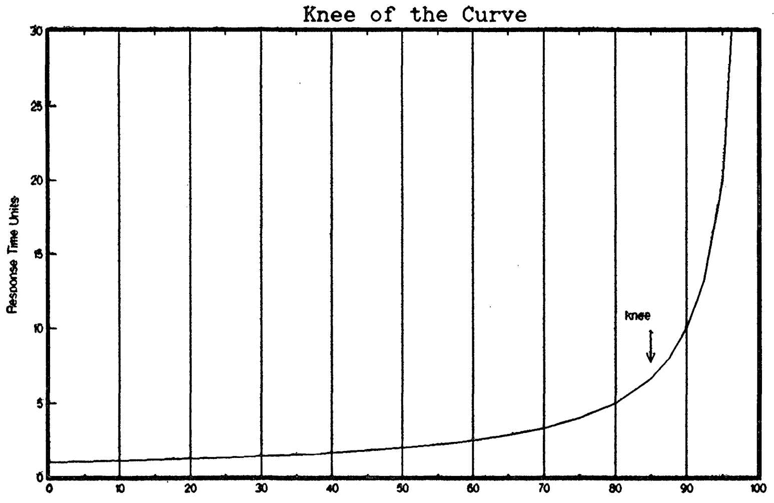


Figure 4

Processor Utilization Percentage
CSY Performance Tools Support

RESPONSE TIME/UTILIZATION RELATIONSHIP

M/G/1 Model with First Come First Serve Queueing Discipline

Response vs
Utilization

Disc caching will add a small amount of system overhead to your system in terms of CPU. Although the actual overhead is relatively small, it can cause dramatic response time increases if the increase puts your total CPU utilization over a magical number that I will call the "Knee of the Curve". Figure 4 shows a graph of the knee of the curve concept. There is a point all systems can reach where a small increase in CPU utilization will cause a very large increase in response time. The actual CPU utilization figure will vary with the individual processor and the application mix running on the system. If a system was running very near the knee of the curve prior to installing disc caching, the Disc Caching overhead could cause enough additional CPU utilization to push the system over the knee of the curve and into dramatically longer response times. A system could also be pushed over this knee of the curve if any additional system load was added either along with Disc Caching or independently. Summary Disc caching has been an excellent product for the HP3000 family. It is modest in its use of system resources and is easy to use. In most environments Disc Caching has given its users a strong price/performance boost. References J. R. Busch and A. J Kondoff, "Disc Caching in the System Processing Units of the HP3000 Family of Computers," HP Journal, February 1985.

J. R. Busch, "The MPE IV Kernel: History, Structure, and Strategies," Proceedings of the HP3000 Internals Users Group Conference, Orlando, April 27, 1982.

J. R. Busch and A. J. Kondoff, "MPE Disc Cache: In Perspective," Proceedings of the HP3000 International Users Group Conference, Edinburgh, October 1983.

"Series 64 with Disc Caching Beats IBM 3033 in Batch MRP Run," Hewlett-Packard Computer News, October 1, 1983.

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik, Quantitative System Performance, Perntice-Hall Englewook Cliffs, New Jersey, 1984.

MPE V Tables Manual, Hewlett Packard Company, 1984.

3069. TRANSACT AND 3RD PARTY SOFTWARE TOOLS USED IN A LARGE ON-LINE ENVIRONMENT

Lisa Wilhelm
Stan Lukoff

E.I. DuPont de Nemours and Company, Inc.
Chemicals and Pigments Department

SUMMARY

This paper is a case study of the development effort of the DuPont Chemicals and Pigments Department 3MCS (M aintenance M anagement and M aterial C ontrol S ystem) project which began a few years ago. A brief overview will be given on the key elements of the system (project backround, module descriptions, data base overview, number of screens, jobs and programs) and then all of the details relating to the use of TRANSACT and 3rd party software will be discussed.

A major decision was made in the use of the TRANSACT language for the DuPont 3MCS application. At the time, the language had just been introduced by Hewlett Packard and there were very few people with TRANSACT experience. Therefore, much work had to be done to develop programming standards for 3MCS. System performance also became an issue when more users were active in the application, and our experiences to date will be shared.

Due to our aggressive development schedule, there were incentives to be as productive as possible. We found that we could fill most of our needs with 3rd party software packages which supplement the HP software tools and MPE operating system. All of the packages we are using will be discussed along with specific benefits. These packages include ADAGER, DBPLUS, FASTRAN, MPEX, OCS, RADAR, ROBOT, and SCOMPARE/COMPARE. We are currently evaluating VTEST and PreVIEW.

The 3MCS System was also designed to be a general system that could be used by many plant sites. Each plant would be able to tailor any special needs by means of a specifications data set in our tables data base. This way, plant customization would be eliminated, and maintenance would be performed centrally by the Wilmington 3MCS support team.

BACKROUND/SYSTEM OVERVIEW

The Maintenance Management and Material Control System (hereafter referred to as 3MCS) was initiated by plant requests for an integrated solution for improving material control and maintenance practices at plant sites. 3MCS is a collection of on-line integrated TRANSACT programs comprising 8 modules.

The objective of the 3MCS system is to reduce maintenance and stores costs by:

- o Reducing stores inventories,
- o Increasing maintenance productivity, and
- o Reducing process downtime.

The original design for 3MCS was to customize Hewlett Packard's Materials Management (MM/3000) package. Customizing MM/3000 was supposed to give us 75% of the needed functionality. We would then develop in-house the 25% functionality not achievable through customizing MM/3000. We decided to use the TRANSACT language to do this since most of the functionality developed in-house would be on-line inquiries. After about six months of developing 3MCS using this dual approach, we became convinced that the design would not work. We realized that customizing MM/3000 would only give us about 50% of the needed functionality and using 3MCS as it ran under this dual approach was extremely cumbersome for the users. We decided to abandon the dual approach and develop 3MCS completely in-house.

The decision was made to write 3MCS completely in TRANSACT. We felt that we could develop 3MCS quicker using TRANSACT than using other languages such as COBOL. We also felt that we could develop a standard system more easily with TRANSACT because TRANSACT handles the calls to the IMAGE and VPLUS intrinsics. We knew that TRANSACT could call programs written in other languages. We figured that we could always write programs in other languages to be called by TRANSACT if a situation arose that we couldn't handle directly with TRANSACT. Using TRANSACT did allow us to develop 3MCS quickly. To date, we have not encountered any situation that required writing programs in other languages. The major drawback, performance, had been discovered when we started running batch jobs. Our solution to the TRANSACT performance deficiencies was to use FASTRAN. Without FASTRAN, we would have had to use a compiled language, such as COBOL for our batch programs.

Along with TRANSACT, we decided to make heavy use of the DICTIONARY. All data bases, screens and common variables are defined in the DICTIONARY. Documentation for these items also exists in the DICTIONARY. Using the DICTIONARY eliminates the need to define items in every program. Changing an item becomes a matter of changing it in the DICTIONARY and recompiling all the TRANSACT programs that refer to it. Additionally, the DICTIONARY provides information for INFORM and REPORT users.

The functions of 3MCS are explained in the following module descriptions.

CATALOG MODULE

The catalog module contains descriptions of equipment pieces used at a plant, and descriptions of spare parts and mill supplies available from the storeroom.

Standard description formats are used including key noun, descriptive adjective, manufacturer, material of construction, and size. The equipment pieces are cross-referenced to common equipment pieces and spare parts.

The objective of the catalog module is to provide easy access to equipment, spare parts, and mill supplies. This allows quick identification of the correct part or mill supply, inventory availability, and an easy method to request material from the storeroom. Rapid identification of material requirements is achieved through a variety of on-line searches. Periodically, batch reports (Catalog Reports) containing equipment, spare parts, cross-references, and mill supply information are printed.

STORES MODULE

The stores module deals with spare parts and supplies; setting up items in inventory, maintaining descriptions and accurate inventories; and receiving, ordering and issuing of items. The module was designed to easily maintain and search for information on-line for these activities. All of the following functions can be performed:

- o Create and print stores tickets, issue items, or process return to stock,
- o Order stores items, receive orders, reverse incorrect receipts,
- o Perform physical inventory fuctions,
- o Review status of inventory levels, order status and ticket requests,
- o Transfer of salvage material, and
- o Adjust quantities, price or value for selected items.

Batch reports are run to provide auditing and accounting controls. Important accounting functions are performed within the module such as updating work orders with material cost and passing the monthly stores material costs to the Corporate Accounting System. One of the obtainable objectives of the module was to achieve service and inventory accuracy in excess of 95% while minimizing inventory.

PURCHASING MODULE

The purchasing module provides easy access and maintenance capability for purchasing related functions. The major functions include:

- o Setting up and maintaining vendor information,
- o Creating stores, non-stores, and emergency orders,
- o Altering orders,
- o Reviewing orders, invoices, and vendor histories,
- o Expediting orders,
- o Closing orders, and

- o Verifying invoices.

A variety of reports are printed for management and accounting control.

WORK ORDER MODULE

The work order module allows users to estimate, create and issue new work orders and projects; and to review, monitor, and revise existing work orders and projects.

Labor and material costs are collected and written to the work order data base. On-line inquiries and monthly reports are generated to inform users of the status of work orders.

Equipment history is updated when a work order is completed and is available via inquiries and reports.

The primary objectives of this module are to:

- o Allow maintenance personnel to improve their handling of work orders, reduce paper work, ease the task of retrieving work order data, and
- o Allow for easy retrieval of work order cost on-line to compare with estimated cost thereby improving the control of maintenance cost.

PLANNING AND SCHEDULING MODULE

The objectives of the planning and scheduling module are as follows:

- o Provide the tools to assist planners and crew supervisors with the scheduling of maintenance tasks,
- o Use priorities and estimates to generate backlog reports,
- o Help planners define resources and assess their availability, and

- o Prepare planners for utilizing more sophisticated scheduling when that complexity is needed to efficiently schedule work.

The planning and scheduling module uses information entered via the Work Order module to provide backlog reports. Daily work schedules for crews are prepared on-line making use of existing entered data (work orders, priorities, due dates, etc.).

The actual hours worked by an indvidual is entered into the module and updates are made to the schedules and work orders.

A weekly batch job is run which summarizes work order completion and scheduling performance. On-line review of this information is available by area and crew.

PREVENTIVE MAINTENANCE MODULE

Preventive maintenance (PM) is the routine overhaul of equipment on a regularly scheduled basis for the purpose of inspection and replacement of worn parts. This type of activity is usually done as often as once per month, with the exception of lubrication activities.

This module provides the tools required to facilitate scheduling of preventive maintenance activities. It also helps the users gather and store equipment history data gained while performing such activity. Finally, it provides tools for quick, easy retrieval for this data.

The system uses information entered via this module and the Work Order Module to generate preventive maintenance schedules, overdue, exception, and performance reports. The schedules list preventive maintenance jobs and their respective due dates. Overdue reports list PM work which is overdue. Exception reports provide information useful in determining the correct work frequency. Performance reports provide feedback on the amount of PM work suggested and the percentage of work completed.

Information gained while doing PM jobs is entered in the data base via on-line interactive CRT's and becomes an integral part of the equipment history files.

Data retrieval is accomplished via the equipment history portion of the Work Order Module and the performance and exception reports of the PM Module.

The objectives of the module are:

- o Provide tools required for scheduling preventive maintenance,
- o Provide for storage of preventive maintenance instruc- tions and history data gained through the performance of preventive maintenance activities,
- o Enable maintenance management to make better use of data gained from a PM program by integrating it with other data, so as to facilitate analysis and possible correcting of recurring failures, and
- o To enable management to adjust, add or delete this type of activity by providing easy retrieval and analysis of history data.

PREDICTIVE MAINTENANCE MODULE

Predictive maintenance is the periodic monitoring of physical variables so as to predict, and thus prevent, costly and disruptive equipment failures. Some examples of predictive maintenance monitoring variables are: vibration, noise, temperature, pressure, flow, etc.

This module helps users schedule predictive maintenance activities. It helps them gather and store equipment history data gained while performing this activity. Finally, it provides them with a tool for quick and easy retrieval of this data.

The system uses information entered via this module, the Equipment Catalog and Work Order module to generate predictive maintenance schedules and exception reports. Information gained while doing this type maintenance is entered in the data base via on-line screens and becomes an integral part of the history files. Alternatives to manually entering the data are being pursued. These involve using hand-held monitoring devices to accumulate various data.

ADMINISTRATIVE MODULE

Some 3MCS functions are common to or a prerequisite for other modules to work. These functions are administrative in nature and include:

- o System, screen, and field security administration and maintenance,
- o Ability to call a new screen via our "DRIVER" program,
- o Administration and maintenance of validity tables,
- o Ability to select and maintain local plant options (i.e. plant name, address, etc.),
- o System control and clean-up jobs such as daily start- up routines,
- o Log file utilities to enable/disable logging and create log file audit reports,
- o 3MCS recovery and restart procedures, and
- o Copy routines to copy the on-line data base to the batch account for month-end processing.

Before any module can be installed, the users of the module must be uniquely identified and screen security determined. Tables must be set up with valid data and plant options identified. These requirements are entered in the 3MCS system via screens in the administrative module.

HARDWARE AND SOFTWARE

HARDWARE

The typical plant will require a dedicated HP/3000 Model 68 computer with 8 megabytes of memory, 6 disk drives (404Mb), 1 high speed tape drive, and a 1000 LPM line printer. Terminal and remote printers vary in model and numbers required for each plant. This configuration should support up to 100 on-line active users, and a stores inventory of approximately 20-25,000 items. The number of disk drives will vary depending on the size of the plant and the amount of data needing to be stored.

SOFTWARE

3MCS was developed using the following software tools; Also listed are operation software tools used by users, operations, and support personnel.

HP SOFTWARE

TRANSACT	Programming language
VPLUS	Screen handling
IMAGE	Data base access software
DICTIONARY	Data dictionary
INFORM	Inquiry language (tool for users)
REPORT	Report language (tool for users)
TDP	Source code editor

(Note: some sites also are using HPLIST and HPMENU)

3RD PARTY SOFTWARE (refer to appendix for more details)

ADAGER	Data base maintenance software
COMPARE	Compares data files to identify differences
DBPLUS	Data base utility that extracts data into flat files
FASTRAN	Recompiles TRANSACT source code into SPL and object code to improve performance
MPEX,STREAMX	Extention of MPE utility for filesets, and job streaming
OCS	Used for operations scheduling of jobs
PreVIEW	Allows use of non-block-mode terminals for 3MCS (i.e DEC terminals),
RADAR	Monitors performance and jobs; sends messages to block mode users (currently inactive)
ROBOT	Cross reference of where elements are used in programs, jobstreams
SCOMPARE	Compares source programs or jobstreams to identify differences
VTEST	Runs interactive screens in batch mode to facilitate testing (curently inactive)

DATA BASE OVERVIEW

3MCS uses the HP IMAGE and DICTIONARY data management tools to define and structure the data. There are currently 5 data bases in the most current release of 3MCS. An additional data base for Preventive Maintenance will be added in a future release. The data bases are:

- STORES data base - contains item descriptions, inventory, stores tickets, stock activity.

  Number of master sets = 14 Number of detail sets = 5 Number of elements = 235 Typical size = 517,000 sectors

- PURCHASING data base - contains item history, vendor data, purchasing data, receiving and invoicing data.

  Number of master sets = 11 Number of detail sets = 15 Number of elements = 200 Typical size = 319,000 sectors

- WORK ORDER and EQUIPMENT data base - contains equipment descriptions, cross-references, work order information.

  Number of master sets = 10 Number of detail sets = 9 Number of elements = 179 Typical size = 218,000 sectors

- PLANNING and SCHEDULING data base - contains daily work schedules, crews, work order backlogs, performance data.

  Number of master sets = 10 Number of detail sets = 8 Number of elements = 95 Typical size = 70,000 sectors

- PREVENTIVE MAINTENANCE data base - PM schedule information, equipment history data, and performance report data.

  Number of master sets = 14 Number of detail sets = 10 Number of elements = 140 Typical size = 150,000 sectors

- o TABLE data base - cost codes, standard description data, plant specific data, security data.

  Number of master sets = 34 Number of detail sets = 6 Number of elements = 173 Typical size = 10,000 sectors

MISCELLANEOUS INFORMATION

Number of VPLUS screens = 380 (including child forms)

Number of Batch jobs = 200

Number of On-line pgms = 200

Number of Batch pgms = 100

Approximate lines of code (on-line and batch) = 300,000

3MCS DESIGN

User requirements played a key role in the design of 3MCS screens and programming standards. The user requirements included:

- o Saving values for key fields and displaying these values as the user moved between screens,
- o Filling fields with back slashes (\) to clear them,
- o Displaying numeric fields with decimals as two separate fields on the screen with the decimal point between the fields,
- o Displaying stars (*) in secured fields,
- o Going from screen to screen without traveling through the menu subsystem, and
- o Using function keys (f1-f8) instead of the enter key.

These requirements, coupled with the desire to make 3MCS as "user friendly" as possible, influenced many 3MCS design decisions. Our design decisions included user requirements, screen standards, and programming standards. At the time, the costs of implementing the decisions were not well understood. From a systems viewpoint, these decisions were grouped into three categories - good, regrettable, and neutral. Some of the reasons for these groupings are explained below.

GOOD DECISIONS

We decided that we would implement the on-line portion of 3MCS as a series of callable TRANSACT programs. Each 3MCS screen would be implemented as one TRANSACT program. We developed a main TRANSACT program that calls all other TRANSACT programs. This main TRANSACT program, DRIVER, uses a table to determine which program to call. The table is keyed by screen name, so that each screen in 3MCS is given a unique name. The name appears in the upper right hand corner of the screen. The user indicates which screen he wishes to go to by pressing a function key or by keying the name of the screen in an area known as the command window. The "DRIVER" approach allows many people to develop screens at the same time. This approach also allows the programmer to write the screen program independently and 'hook' it into the system at a later date.

There is data that needs to be shared between DRIVER and the program it's calling. This data must be placed first in the TRANSACT LIST and DATA registers in the two programs and it also must be mapped identically in both programs. We use a TRANSACT include file, LCOMMON, to accomplish this objective. LCOMMON contains the LIST command for the shared data items. We established a programming standard for screen programs - nothing is to be placed in the LIST and DATA registers ahead of the items in LCOMMON. We also use TRANSACT include files for common routines needed by many screen programs. Using TRANSACT include files allows changes to be made quickly and easily and provides an easy way to standardize the system.

Messages are used to direct the user through the operation of the screen or to explain errors. We decided to use a "message catalog" for the on-line portion of 3MCS. Every message is given a unique identifier. The identifiers and text of the messages are stored in a manual master. Each screen program uses a common routine to retrieve the message just prior to displaying the message to the user. This practice lets programmers change messages without having to recompile programs. Not having the text of the message in the programs saved stack space. Having 3MCS messages in a manual master made it easy to get listings of the messages. Listings of the 3MCS messages made it easy to standardize their wordings and format.

NEUTRAL DECISIONS

We decided to use one forms(VPLUS) file. Using one forms file eliminates overhead associated with a second forms file. The forms file is opened initially and closed when the user exits 3MCS. But 3MCS screens are diverse. Having all the 3MCS screens in one forms file means that TRANSACT requires more stack space to process the forms in the forms file. Therefore less stack space is available for TRANSACT to use for other purposes.

We decided not to use VPLUS edit specifications. The primary reason for this was related to "user friendliness". 3MCS reads each screen whenever a user presses any function key. Some of the function keys cause the user to be transferred from the screen that he is on to a different screen. With VPLUS edit specifications, if the user keyed incorrect data into fields and then pressed a transfer key, he would have to correct the data before he could go to the new screen. This was considered unfriendly. We also felt that the system would be easier to maintain if all validity was done in the actual program. We could not make use of our message catalog with VPLUS edit specifications, and having VPLUS edit specifications would potentially increase the amount of stack space TRANSACT requires to process the forms file.

We decided to make most screens multi-functional. For example, a user would go to a single screen to review, add, change or delete an item. We chose to implement this design using parent-child forms. This decision was good for the user because it reduced the total number of 3MCS screens. But the screen programs themselves were larger. Many of them had to be segmented to work.

We also decided to use different variables for the fields on the screen and the fields in the data bases, even when they represented the same item. This practice made the TRANSACT programs easier to maintain. A programmer did not have to worry about whether the value of the variable was the 'old' value from the data base or the 'new' value from the screen at any given time in the TRANSACT program. However, each program had to have routines to move the values in the screen variables to the data base variables. This often involved data conversion.

We decided to use an "X-option" in several 3MCS screens. The "X-option" can be described as the user's ability to select items from a list by placing an X in a designated spot next to the item. The list of items to be selected is usually the result of an on-line search. Although the "X-option" is very user friendly, the TRANSACT screen programs that use this option are difficult to maintain.

Another decision was to stay entirely in block mode. By staying in block mode, we eliminated the overhead associated with switching between block mode and character mode. This also eliminated potential confusion between the ENTER and RETURN keys. However, users were not able to use the roll backwards and forwards keys in search screens because we stayed in block mode.

REGRETTABLE DECISIONS

We decided to make all VPLUS fields type CHAR and to bypass all VPLUS field data verification. This was especially burdensome for numeric fields. We were not able to take advantage of numeric checks available through VPLUS. Instead we had to write our own routines to do numeric checking. These routines had to examine each character in the field to make sure that a digit (0-9) had been keyed. These routines had to handle signed numbers and numbers that had been keyed anywhere within the field. The routines lengthened the screen programs causing more stack space to be used. More data conversions had to be done as well.

For similar reasons, the decision to split numeric fields with decimal values into two separate fields on the screens was also poor. Complex routines to display data in these fields and to accept data from these fields are required.

GENERAL OBSERVATIONS

We learned a great deal about TRANSACT as we used it to develop 3MCS. One of the challenges 3MCS programmers continue to face is how to overcome stack limitations. Many screen programs must be segmented. The segmented programs must be retested when new versions of TRANSACT and MPE are installed. The new TRANSACT/MPE versions may use more stack space. Programs that ran before the version upgrade, may not run after. Changes to the DRIVER program must be made carefully and in a way that minimizes the amount of additional stack space required.

Programmers need to code TRANSACT programs "procedurally". By "procedurally" we mean using PERFORM statements and being careful with GO TO statements. This becomes especially important if a program needs to be segmented. The TRANSACT statements that make up a segment must be physically together in the TRANSACT source code. A program written "sequentially" may be difficult to segment. It is also helpful to write common routines for error handling, screen I/O, etc. to make programs function the same and ease future program maintenance.

An important TRANSACT technique to learn is to use the LIST and DATA registers dynamically. Typically programmers learning TRANSACT will list all the variables they plan to use at the start of the TRANSACT program. TRANSACT treats the LIST and DATA registers as a stack. The item at the top of the list is the last item that was listed. Whenever a variable is referred to, TRANSACT begins at the top of the list and searches backwards until it finds the variable. Keeping the LIST and DATA registers short reduces the amount of time TRANSACT spends searching the LIST and DATA registers.

TRANSACT test modes were very useful for debugging and fine tuning programs. We found the following modes to be the most useful:

Mode 2 - useful for tracing the actual statements executed during program execution

Mode 4 - useful for fine-tuning programs; gives a clear indication when segments are swapped and can be used to evaluate various combinations of statements with respect to timing

Mode 25 - useful when you are having a problem with IMAGE data bases

Mode 34 - useful when you are having a problem with VPLUS screens

Mode 43 - useful when you are having a problem with your LIST and DATA register; should be used with discretion because this mode can produce volumes of paper

Mode 102 - useful in fine-tuning the DATA and WORK options of the SYSTEM statement

SEGMENTATION AND STACK CONSIDERATIONS

STACK LIMITATIONS

o Present stack limitation is slightly in excess of 32,000 words.

o 3MCS currently requires about 7500 words for our forms file,the PCBX, and the transact outer block areas.

o The DRIVER program currently requires about 4600 words.

o Allowing an area of 8000 words for system called routines & expansion this leaves about 12,000 words as our maximum program size.

o Any program which has the final stack usage near 12,000 words should be considered for segmenting during the next modification.

o Using the "OPT" option on all items defined in your program which are not used in LIST= constructs,display item headings, or prompt strings can significantly reduce your stack.

SEGMENTING RECOMMENDATIONS

o Always create the root and at least 2 other segments.

o Put infrequently used code in a separate segment ie. error rtns. etc.

o Put "shared data" INCLUDE files in the root segment and keep the root as small as possible.

o If possible put code frequently used by more the one segment in the root. ie. GET(FORM) etc.

o Always leave/enter a segment with a GO TO or PERFORM. Program will not just fall thru to the next segment in your listing.

o Parent/child relations defined in the dictionary having the parent only referenced in the root may not recognize the child in a later segment.

o Minimize segment transfers. Run your program in test mode 4 to determine the number of segment transfers as well as the stack used by the program. Stack= Z + D + 1500.

o Watch for FIND with a PERFORM = where the PERFORM = is in another segment because this will cause two transfers for each record read.

o Try to create segments by logical function.

o Any items added to the list,match or update registers must be removed before exiting that segment.

o The GO TO statement referenced by the "ERROR=" option on data base access verbs must reside in the same segment.

o As a "temporary" last resort the SWAP option can be used by specifying it in the DRIVER program. This saves about 2300 words.

o For those users which experience the intermittent stack overflow problem because they open many datasets by accessing many 3MCS screens, a new user logon should be used which would use the NOCB option when running TRANSACT.

CONCLUSIONS

The 3MCS application is large and complex, and has been successfully written using TRANSACT, IMAGE, and VPLUS. TRANSACT is a powerful, flexible language that allows programmers to develop applications quickly and easily. Knowledge of VPLUS and IMAGE is essential to be able to use TRANSACT efficiently for applications that use VPLUS screens and IMAGE data bases. Using TRANSACT is not a substitute for doing good design. Design decisions should be made carefully by evaluating the full costs and benefits associated with using the design.

APPENDIX

3rd Party Software Description

Software Package:	ADAGER
Vendor:	ADAGER APARTADO 248 ANTIGUA, GUATEMALA
Contact:	Alfredo Rego (502-2) 324336 Telex 4192 Teltro GU

Where Needed:

3MCS development/maintenance site (Wilm. only) and sites where 3MCS is running in production.

Product Description:

ADAGER is a data base transformation tool for the HP 3000. It allows you to dynamically alter the structure of an IMAGE 3000 data base.

Reason Needed:

With the increased size of data base files and applications, issues of performance (speed and reliability) have become very important. Characteristics such as capacity, blocking factor, disc address, and number of sort items have become a crucial part of data base design, and the chances of getting them right the first time may be slim. Unfortunately, the methods provided by the HP IMAGE utilities for adjusting these characteristics and for changing other structural qualities of the data base are often slow and cumbersome. To make a minor alteration, such as changing the name of a data set or raising the capacity of a data set, requires that you unload the data base to tape or disc, purge it, modify and recompile the dictionary or schema, build a new data base, and finally reload the data from tape or disc. If the bases are large, as is the case with 3MCS, this process can take 10-16 hours.

Benefits:

ADAGER has saved countless hours of computer and programmer time by allowing capacities to be dynamically changed without an unload/reload.

3rd Party Software Description

Software Package: COMPARE

Vendor: ALDON COMPUTER GROUP FINANCIAL CENTER BUILDING 405 14TH STREET 90069 OAKLAND, CA 94612

Contact: JOAN S. BRODSKY 415-839-3535

Where Needed: 3MCS development/maintenance site (Wilm. only)

Product Description:

Comparison of output data files has always been a vital part of testing the accuracy of changes made to programs. In most cases this is a tedious, manual job performed by programmers looking at printed output. COMPARE allows the computer to perform that function with with either MPE, KSAM, or IMAGE files. Differences are pinpointed with the specific positions in the record that are different highlighted for rapid identification. Various options are available such as comparing records of different lengths, ignoring fields that are expected to be different and comparing for equality.

Reason Needed:

This software is needed to aid in 3MCS software testing and is not available from HP. COMPARE will eventually will be used as a data integrity check for new releases.

Benefits:

Programmer testing time is maximized and not wasted by looking at every field in a file.

3rd Party Software Description

Software Package: DBPLUS

Vendor: SOFTWARE DESIGN ASSOCIATES 11260 ROGER BACON DRIVE SUITE 300 RESTON, VA 22090

Contact: CURT LOUGHIN/RON KAHLOW 703-471-0076

Where Needed: Sites where 3MCS is runnng in production and 3MCS development/maintenance site

Product Description:

DBPLUS is an IMAGE/3000 data base productivity software whose strength is its ability to manipulate the actual data in a data base. It performs 3 basic functions:

- o A LOAD function can load or update data sets from tape or disc files,
- o An UNLOAD function can unload data sets to tape or disc files, and
- o A COPY function can load or update dat sets from data contained in one or several data sets.

The power of DBPLUS comes from a variety of situations and problems these basic functions can be applied to. A wide range of options permit reformatting data, changing data types, linking and building record complexes, and applying selection criteria.

Reason Needed:

This software is needed to create MPE files at month-end for batch jobs. The batch jobs run much faster using an MPE file as compared to reading a data base. The utility features are also used for special applications requiring fast response.

Benefits:

This software package eliminates writing TRANSACT programs to do data base manipulation. Maintenance is very easy to perform when DBPLUS is used instead of a program. Benchmarks show that DBPLUS is faster than compiled COBOL in performing its functions.

3rd Party Software Description

Software Package: FASTRAN

Vendor: PERFORMANCE SOFTWARE GROUP P.O. BOX 1464 SANDY SPRING, MD 20860

Contact: HOLLY SILER/NICK DEMOS 301-977-1899

CURT LOUGHIN 703-471-0076

Where Needed: 3MCS development/maintenance site (Wilm. only)

Product Description:

FASTRAN is a compiler for TRANSACT source programs. It produces SPL code that is then processed by the SPL compiler. The object program can then be prepared and run like any other program. The major advantage is a dramatic decrease in CPU time as well as response time. FASTRAN compiled programs also use less data stack space and less overall system memory, particularly if the same program is being run by several terminals.

Reason Needed:

Inadequate performance of TRANSACT processor along with need to run more terminals on an HP CPU.

Benefits:

Elapsed times of batch jobs have run 2 to 10 times faster after compiling in FASTRAN. CPU times are reduced about the same ratio as elapsed times. On-line testing has shown CPU utilization to be reduced by 30-50% along with the same reduction in stack space. FASTRAN will reduce stack space by eliminating the intermediate processor code which TRANSACT maintains on the process's stack. FASTRAN also eliminates time consuming operations such as list register searches, table look-ups when child items are referenced, and also employs COBOLII microcode instructions wihic greatly enhance certain functions.

3rd Party Software Description

Software Package:	MPEX
Vendor:	VESOFT, INC. 9213 WARBLER PLACE LOS ANGELES, CA 90069
Contact:	EUGENE VOLOKH 213-859-9666
Where Needed:	3MCS development/maintenance site (Wilm. only) and sites where 3MCS is running in production

Product Description:

MPEX is a useful productivity and system management control tool for any HP/3000. MPEX enables handling of filesets (data bases, KSAM, MPE) in all MPE commands. This enables a user to do things like FCOPY @.SOURCE or PURGE @.DATA. The fileset concept itself is also extended: files can be selected not just by name, but by CODE, CREATOR, DEVICE, ACCDATE and other parameters. Every MPEX command can be executed from EDITOR or TDP (or QEDIT or QUAD). MPEX also allows for very sophisticated handling of mass compiles by the USER command which executes a template file.

Reason Needed:

The limitations of the HP MPE operating system necessitated using MPEX to gain vast productivity improvements for programmers as well as system management functions. With all of our account manipulation, MPEX has been a real bargain and takes the drudgery out of fileset manipulation.

Benefits:

MPEX has allowed us to realize much higher programmer productivity due to the time davings in executing commands on filesets. It has also been extensively used in account merges since program development is performed in more than one account.

3rd Party Software Description

Software Package: OCS

Vendor: OPERATIONS CONTROL SYSTEMS 560 SAN ANTONIO ROAD PALO ALTO, CA 94306

Contact: ALLAN MARCUS 415-493-4122

Where Needed: 3MCS development/maintenance site (Wilm. only) and sites where 3MCS is running in production

Product Description:

OCS is a totally integrated software package designed to increase substantially the productivity of the HP/3000 data center. OCS automates multiple-queue job dispatching, production scheduling, forecasting, tape library management, resource utilization, data center accounting, jobstream maintenence, and security.

The OCS Dispatcher runs any number of interrelated jobs concurrently with full dependency control. The Dispatcher initiates, tracks, and controls batch production. It maintains job sequencing and priorities and logs individual job termination status. The Dispatcher's online screen facility prompts operators for required procedures and allows them to check current job status, insert special jobs, and issue commands while OCS/Dispatcher continues launching jobs and tracking production in the backkround.

Reason Needed:

HP does not currently have software to perform these necessary operations functions. OCS provides greater reliability, fewer mistakes, better service, shorter total processing times, and greater control over the data center.

OCS allows for standard production networks to be developed for daily, weekly, and monthly jobs.

Benefits:

The daily jobs total run time has been reduced by 30-45 minutes per day. This was time where the users were normally waiting to log on to 3MCS. The month-end network of jobs was reduced by 3 hours after using OCS. In addition, operator productivity has been increased since they do not have to keep track of each job's status.

3rd Party Software Description

Software Package: PreVIEW

Vendor: TYMLABS CORPORATION 211 EAST 7TH STREET AUSTIN, TEXAS 78701

Contact: TERESA L. NORMAN 512-478-0611

Where Needed: Sites where 3MCS is running in production and non-block terminals are desired.

Product Description:

PREVIEW allows you to run VPLUS applications on any CRT or PC without any changes to on-line programs or forms file. Wtih PREVIEW, screens look the same, and applications work the same as they do on any block mode terminal.

Reason Needed:

This software allows non block mode terminals to run 3MCS without having to have 2 vendors terminals on a users desk.

Benefits:

PreVIEW is still in the process of being evaluated for 3MCS sites that will be have non-block mode terminals on the plant.

3rd Party Software Description

Software Package: RADAR

Vendor: COMPUTING CAPABILITIES CORPORATION 465-A FAIRCHILD DRIVE SUITE 122 MOUNTAIN VIEW, CA 94043

Contact: FRANK PINKELA 415-968-7511

Where Needed: 3MCS development/maintenance site (Wilm. only) and sites where 3MCS is running in production

Product Description:

RADAR is a software system which monitors, controls, and collects performance data from any HP/3000 on-line application. RADAR provides system management with previously unavailable data such as terminal utilization, transaction mix and throughput, number of user inputs, and system response times. The monitored programs can also be controlled, using simple commands which allow you to specify the days and hours when a program may be run, and which terminals may use it.

Other useful features include the ability to show the active terminals and send messages to terminals in block mode without overwriting user data on the screen.

Reason Needed:

RADAR was at the time the only way to collect certain statistics on the 3MCS user profile. It also allowed for collecting transaction data for various 3MCS users to see what user response was under various computer load conditions.

Benefits:

RADAR is a good way to monitor increased user activity as well as being able to communicate with users in block mode when a system shutdown is pending. Activating RADAR requires no programming and does not affect the applications program or operating procedures.

3rd Party Software Description

Software Package: ROBOT

Vendor: PRODUCTIVE SOFTWARE SYSTEMS 5617 COUNTRYSIDE ROAD EDINA, MN 55436

Contact: ROGER OLSEN 612-920-3256

Where Needed: 3MCS development/maintenance site (Wilm. only)

Product Description:

ROBOT consists of 3 products: Automatic Documentor, Documentor Plus, and the Auditor. The ROBOT Automatic Documentor is a unique system cross-reference tool to analyze the impact of changes on existing source code. In the case of 3MCS, we have defined jobs, source code, and "include" files to be cross-referenced.

The ROBOT Documentor Plus is an interface to the Automatic Documentor data base and the program files. It shows where and how an item is used within a source file or job stream. This way it is easy to see in what context a data element is being used without having to go into the program.

The ROBOT Auditor is a file management tool that shows which files have been changed or restored and when. It also reports any files purged from disc, intentionally or accidently.

Reason Needed:

The HP DICTIONARY does not keep track of any program cross- references. By using ROBOT, we are assured that all occurences of an element have been found, where performing this manually, we would never be sure that something had been overlooked. This software ensures a complete job by the maintenance or support programmer and no surprises later. Estimation of the time for specific enhancements is greatly improved by knowing the number of programs affected.

Benefits:

A quality maintenance job will be assured as well as maximizing programmer productivity. Future maintenance will be done with much greater efficiency.

3rd Party Software Description

Software Package: S/COMPARE

Vendor: ALDON COMPUTER GROUP FINANCIAL CENTER BUILDING 405 14th STREET OAKLAND, CA 94612

Contact: JOAN S. BRODSKY 415-839-3535

Where Needed: 3MCS development/maintenance site (Wilm. only)

Product Description:

SCOMPARE is a unique source comparison tool which can identify clearly the difference between two program source modules. It identifies all inserted, deleted, moved, or changed records.

Reason Needed:

This is a product which HP currently does not offer and establishes "production confidence" with new releases of 3MCS.

Benefits:

Allows us to do our application development integration much more safely, which gives us a much greater assurance that our quality standards have been maintained. SCOMPARE also eliminates the need to do laborious line by line desk checking when determining which version of source code is "correct".

3rd Party Software Description

Software Package: VTEST

Vendor: TYMLABS CORPORATION 211 EAST 7TH STREET AUSTIN, TEXAS 78701

Contact: TERESA L. NORMAN 512-478-0611

Where Needed: 3MCS development/maintenence site (Wilm. only)

Product Description:

VTEST provides automatic, repeatable testing of on-line programs, greatly simplifying the debugging and testing process. Using script files that look like user input, VTEST logs on and runs your test script unattended, just like a user at a terminal. A printed report is produced documenting the entire session, showing the screen image before and after each input. For multi-user environments, VTEST can run simultaneous sessions in batch to thouroghly test file locking procedures. Script files can be saved for easy testing after any change.

Reason Needed:

VTEST allows for thorough "regression" testing when preparing for a new release of 3MCS. It assures that everything works prior to landing in the users hands. It also allows for systematically creating test scripts for key on-line screens.

Benefits:

VTEST is not yet available for MPE V/E and has consequently not been tested with 3MCS.

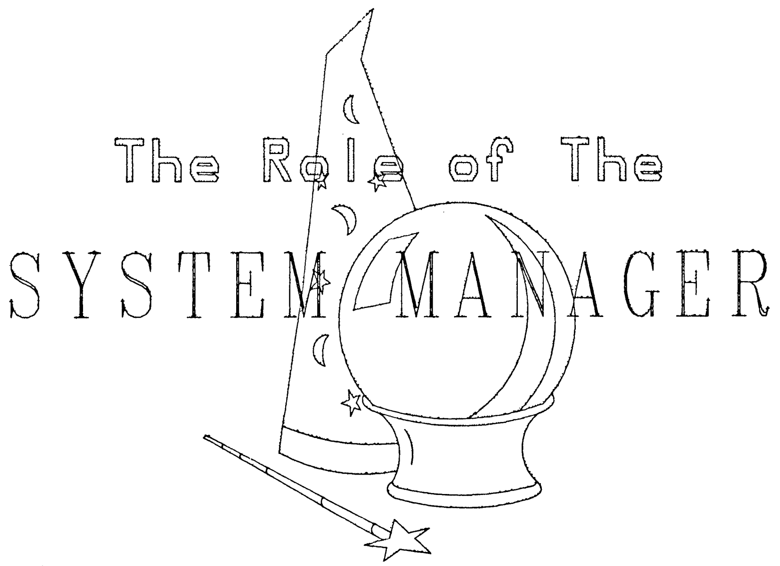
3071. The Role of the System Manager

Tom Idema
Westinghouse Furnature Systems
4300 36th Street S. E.
Grand Rapids, Michigan 49508

Theme:

The System Manager exists: the function is vital and the rewards are real. You have only to recognize these facts and make the proper application of the System Manager's talents to reap an efficient, smooth operating system. Because a computer system requires constant care and maintenance, competent system management is vital to success.

SLIDE #1 :



1. Introduction

Almost every HP3000 has a System Manager, whether called by that title or not. However, the role and responsibilities of the System Manager are quite often misunderstood and their efforts are often wasted or misapplied, resulting in the abuse and or misuse of the system. This paper will address some of these points to show "who", "what" and "how" to properly use the System Manager to ensure a "successful system." Topics covered include what a System Manager is, who it should be, and what they should do, including system monitoring, tuning, planning, standards, etc. It is intended to be of general interest to both the sophisticated as well as the novice HP3000 user.

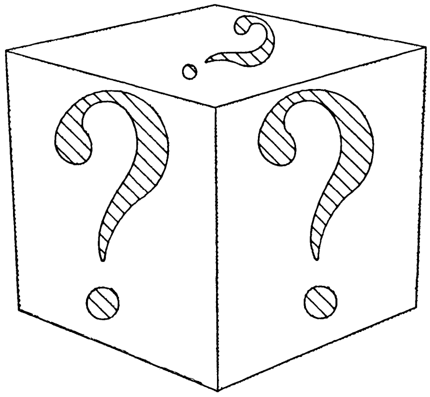
II. The Mechanics of System Management:

SLIDE #2

What is a

SYSTEM

MANAGER

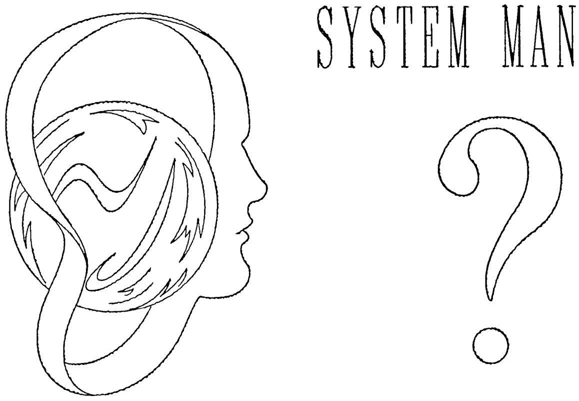


A. What is a System Manager?

The System Manager is both a person and a function (system management), but it is not a function to be performed by just anyone who happens to be handy. Once upon a time, if someone had asked what computer could stand alone, run without an operations staff or systems programmer in an uncontrolled environment, the answer, "An HP3000!" would not have been unusual. Today, however, that simply is no longer true, (unless you'd like to discuss a System 37). The fact is, Hewlett Packard requires that customers with CSS/PICS support have a "trained" individual "designated" as the System Manager, as well as an alternate who is equally trained. The System Manager, therefore, is a trained individual with a specific set of skills, who is responsible for the efficient and effective operation of an HP3000 computer system by maintaining system software and hardware integrity.

SLIDE #3

Who should be the SYSTEM MANAGER ?



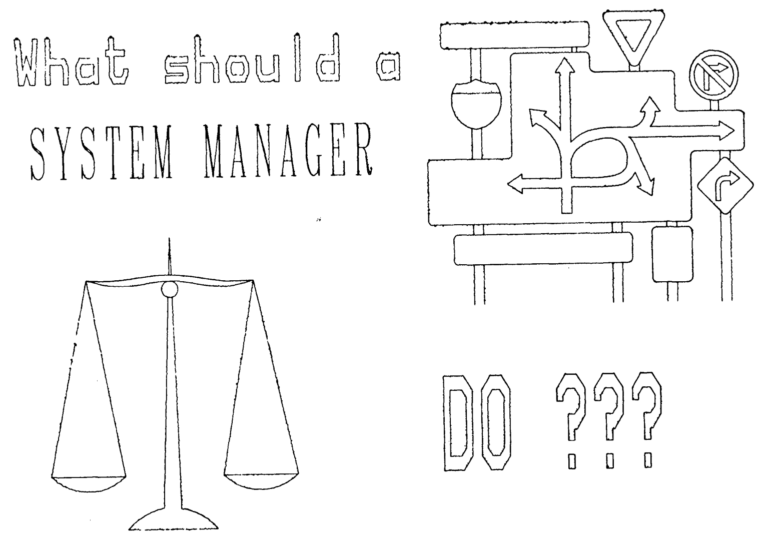
B. Who should the *System Manager be?

Because a System Manager's first responsibility is to the overall system, it is easier to say who the System Manager should NOT be. The System Manager should not be a system programmer, since a systems programmer would be very expensive in the first place and simply not necessary in the second. If in some very remote instance systems programming is needed, these requirements can be provided by, and rightfully belong in, the domain of Hewlett Packard.

Likewise, the System Manager should not be a programmer, an analyst or other member of the systems development staff whose primary responsibilities lie in some other area. This type of person generally has the wrong perspective and priorities to do an effective job of system management. When a programmer is in charge of system management, system operation and performance usually take a back seat to "getting the application system up on schedule."

To be System Manager requires that the individual wear a System Manager's hat and a policeman's badge, each of which are acquired through training and experience. The hat helps keep the System Manager's attention focused on the system's performance and its day to day operation, while the badge helps keep the System Manager's priorities in line with the operational philosophy of the business and allows the enforcement of standards and system requirements. With these things in mind, it is clear that the System Manager has to be someone whose first responsibility is to the system and whose secondary function might be in another area.

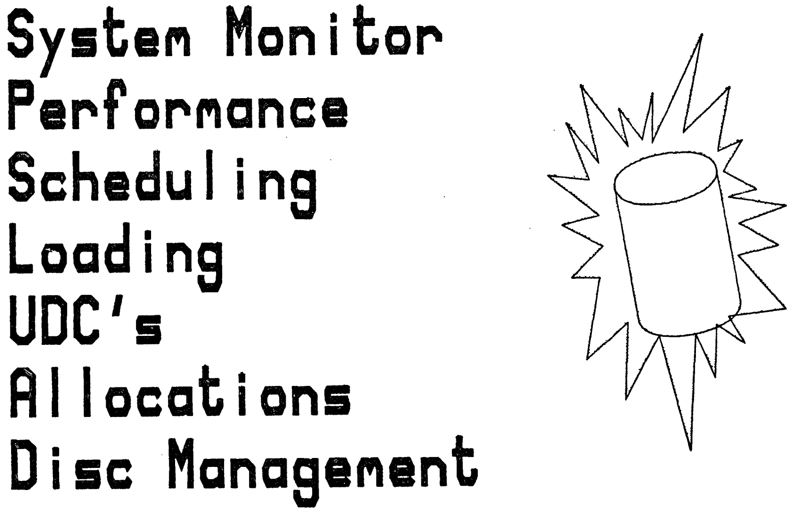
SLIDE #4



What should a SYSTEM MANAGER DO ???

C. What should a System Manager do?

SLIDE #5



System Monitor
Performance
Scheduling
Loading
UDC's
Allocations
Disc Management

System Monitor

One definition says the System Manager's job responsibility is to plan, organize and control the function of the HP3000 so that it best serves the needs of those who depend on it. Although agreeing with this definition, most System Managers would say that they are primarily responsible for the care and feeding of the HP3000 computer system. This is not to say that it is all they do, nor that Hewlett Packard systems require constant attention. However, like most machines, without a certain amount of care they will quickly cease to operate as designed. The System Manager makes sure this doesn't happen by performing a large variety of tasks.

Performance

One of the most important tasks performed by the System Manager is performance monitoring. This involves looking at the overall performance of the equipment over time to determine what is optimum for your installation. There are no set guidelines or recommendations from HP to give you an optimum performance level since every site is, indeed, unique. Performance is a matter of trade offs and the balancing of one parameter against another to develop the best performance according to your user's expectations. When looking at these expectations, the System Manager has to look at the system configuration as a whole, keeping in mind the CPU, memory usage, I/O requirements, disc utilization, etc.

Scheduling

Some items which the System Manager should monitor in regard to performance are scheduling, (priorities), and the mixture of jobs/sessions as they affect response times. Systems are usually characterized as batch or on line depending on a number of factors. Once a system has been characterized as a transaction processor or as a "chruncher," or both, the TUNE command can be used to provide optimum scheduling. While there is no universal setting to provide optimum processing under all conditions, you might want to use different settings at night when processing is primarily batch in nature and another setting during the day which favors the on line users. Just remember, that while a longer setting favors the CPU bound processes, a setting which is too short could require more system overhead and memory manager activities than desired and result in very poor performance.

Loading

System loading should be monitored regularly to determine whether some jobs should be run on another shift or after hours in order to free up prime time for on line access. Since there is a high correlation between system loading and response time, it stands to reason that the better we manage the loading the better we can control response time.

Some of the things the System Manager can do to control the demands on MPE and to even out the load include managing the job/session mix, deferring peripheral equipment loads, (tape and printer), to non critical periods, minimizing data communications sub systems use and ensuring that users employ good habits by not using SHOWJOB's and by not logging on and off of the system excessively. (The system overhead for an idle terminal is quite minimal compared to the overhead incurred at "sign on"). UDC's

In addition to the management of system loading to optimize performance, the System Manager should control the seldom addressed areas of UDC's and program allocations. Although UDC's are convenient and occasionally provide a low grade security, they can cause longer logon and slightly longer command execution time if they are not kept short. More frequently used commands should be placed at the beginning of the UDC file where they can be accesed more quickly, while individual users should not be allowed multiple UDC's, nor should they be set up for users who don't need or won't use them.

Allocations

Program allocation can speed up the :RUN command for an often used program by reducing disc I/O's by as much as 80 to 90 percent. To take advantage of this, however, the System Manager must make certain MPE tables are large enough to accommodate several allocated programs and normal processing as well. These include, among others, the CST and XCST. The savings in performance is usually well worth the expense of increasing table sizes when one considers, for example, that I/O's associated with running the EDITOR are reduced from a normal 156 to only 19!

Disc Management

Fragmentation

Disc management, load and organization have to be administered by the System Manager regularly since bad file placement and fragmentation can literally kill system response time. For instance, whenever the system's disc drives have over one hundred entries in their free space tables before finding a free space segment of over 100 sectors, it is usually time to do a system reload. Although recovering lost disc space will help, a reload will provide tighter file placement and provide better performance. Fragmentation not only hurts performance but can make it impossible to run some jobs or place larger file extents. To create files with multiple extents in a fragmented environment will certainly cause excessive head movement.

File Placement

Poor performance due to bad file placement can be reduced by "placing" KSAM key and KSAM data files on separate disc drives and where two files are used by a single program, they too should be on separate drives. File placement can be controlled with careful use of device classes and readily available utilities such as VESOFT's MPEX.

SYSDISC

If possible, refrain from putting a class of DISC or SPOOL on LDEV1, the SYSDISC. You would be surprised what this will do for performance since access to MPE, the system directory and other system related files is greatly enhanced when the system does not have to compete with operational I/O's on this drive. To keep from wasting unused space on the SYSDISC by using this technique, use it to archive seldom used files and as a source code library.

Caching

Disc caching, if you have both CPU and memory to spare, can significantly reduce system I/O and therefore, improve system performance. In some cases it can reduce disc I/O's 50 percent and more. Although many users do not experience similar improvements, it is usually because they don't have the CPU to spare or that I/O really isn't the problem, per se.

Blocking

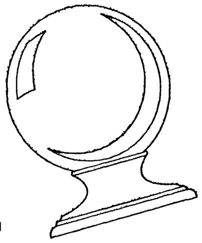
Another thing the System Manager can do with regard to effecting good disc management is to ensure proper use of blocking factors. Very simply stated, have your programmers block as close to sector boundaries, (without exceeding them), as possible since I/O transfers are done by sector. This will help you get the most out of each disc I/O. Sequential files should have larger blocking factors with BUF=2; random files should have smaller blocks with BUF=1. Use of the contributed library program "Block" will help you determine the optimum blocking factors for your files.

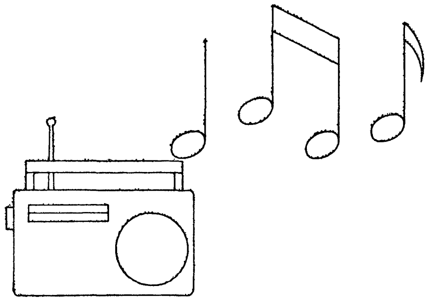
Extents

The conscious use of the file extent allocation process will also affect system performance. While allocating extra extents to accommodate file growth takes time and temporarily wastes space, performance is generally better because the extents are more likely to be contiguous than if allocated on an "as needed" basis.

SLIDE #6

Data base
Query
Application Development
Housekeeping
Tools
Tuning





Data Base

Data base performance is another item which requires monitoring. As a user of disc, a data base which is in need of housekeeping, or is poorly designed, can ruin system performance. It is also a good idea to keep heavily used master and detail data sets on separate disc spindles.

In the area of system development, it is a good idea for the System Manager to monitor the use of IMAGE, data base design and system definition as it affects system performance. A poorly designed data base can often cause degradation which is very difficult to identify and at times even harder to rectify. Performance improvements with IMAGE can also be obtained by

limiting the use of sorted chains, by minimizing the number of paths in the data base to reduce pointer maintenance and the overhead associated with each path and by selecting the most frequently accessed path as the "primary" path since by default the first unsorted path specified in the schema will normally be the primary path.

QUERY

Along with IMAGE, the unchecked use of QUERY by careless and untrained users is another means by which systems performance can be destroyed. When properly used by trained, conscientious users, QUERY can be a most useful and powerful tool; but in the hands of the untrained, it is like a loaded gun in the hands of a child...aimed at the heart of your system.

Application Development

The development habits of the application programmers also have to be policed with regard to system performance. Programs for the HP3000 should be written to take advantage of, and at the same time not abuse, its unique features and capabilities. The System Manager should also look at program stack management techniques and use of system resources or special capabilities being used as a regular part of performance monitoring. Remember, even a System Manager can't make a poorly designed, poorly written application run well.

Other habits a System Manager should make application programmers adhere to include the opening and closing of as few files as possible and leaving files open for the life of a process if they are frequently used. Programmers should also strive for good code locality and intelligent program segmentation because disc access take several times longer than memory access. Swapping, therefore due to bad locality is very costly in terms of system performance.

With experience, one of the things the System Manager will gain is an appreciation for patterns. Patterns of usage, growth and problems will, over time, become vital to the effective management of system performance as will a sense of an "orderly house."

Housekeeping

The orderly house involves nothing more than a periodic dose of "housekeeping," which, if done on a regular basis, will keep many of the performance bugs away from your doorstep. Through housekeeping, which could amount to no more than deleting KEEP or KFILES, old log files and obsolete files, (programmers are wonderful pack rats), performance and space can sometimes be greatly improved. In fact, maybe you won't need that new disc drive after all!

Tools

There are several tools available to the System Manager which make life a little easier. Among these are TUNER, some contributed library programs for data base and program analysis and HP's OPT/3000.

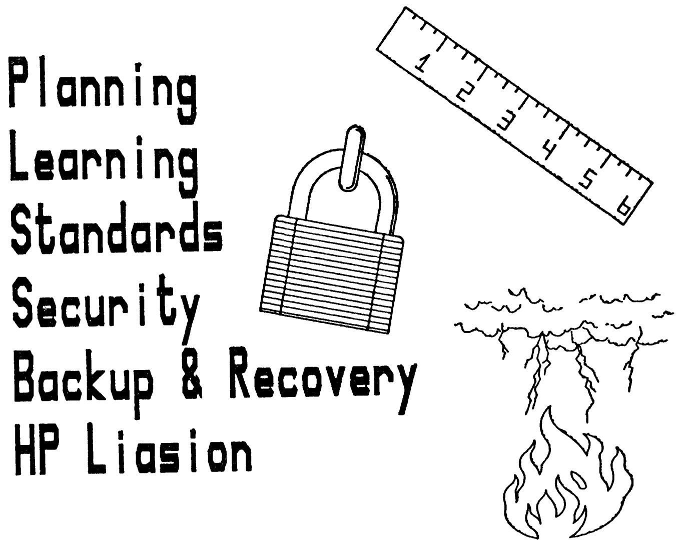
Tuner is a program which displays the system table configurations and their percent of usage and high water marks. With Tuner, the System Manager can determine whether the table configurations are properly sized for the use of that system. The INTEREX Contributed Software Library (CSL) programs PROGSIZE and PROGINFO are helpful tools which assist with program performance analysis. Another CSL program DBLOADNG, is useful for analyzing the load statistics and data base performance characteristics and should be run periodically on all IMAGE data bases.

OPT/3000 is an invaluable tool which allows the System Manager to look deep inside the HP3000 and to monitor almost everything that is going on. It is far superior to any crystal ball, (which is what System Managers used prior to OPT/3000). The purchase of OPT/3000 requires a two week training class which includes over a week of advanced system internals classes. I highly recommend this product to anyone who is serious about system management.

Tuning

Tuning is another of the System Manager's major tasks which is closely related to performance monitoring but deals more directly with the system tables and their configuration. A tool such as OPT/3000 can make life much easier when it comes to monitoring table usage and configurations. If you don't have OPT/3000, Tuner is about the only other tool available for this. A good rule of thumb with regard to table sizing is to configure larger than necessary, if possible. The trade off is real memory table space versus bumping into table limits and possibly suffering a system failure. The space is usually cheaper than a critical system failure in most cases.

SLIDE #7



Planning

Installation planning is another area, which with proper performance monitoring and tuning, a System Manager should also have enough knowledge to do some informed planning for future system growth and performance enhancements. The System Manager, armed with the system data, can specify additions to disc or to memory for effecting improved performance and be relatively certain in that recommendation.

The System Manager should also be the individual consulted by the programming and development staff during the design phase of new systems so that comprehensive hardware and software planning can proceed within both groups and the methodologies approved.

Learning

A System Manager never really knows enough about the system to sit back and not learn more. A System Manager should always be trying to learn as much as possible about the system. One of the ways to do this is to read. System Managers should read the manuals, (in some cases re read them), to be the "resident ex pert" and resource person for the programming staff and be able to talk intelligently with HP or other technical representatives about the system, it's operation or problems.

The COMMUNICATOR, (when there is one), INTERACT and the CHRONICLE are all excellent sources of information which are available to keep the System Manager current on new products and system enhancements. Armed with this knowledge, the System Manager can then take appropriate action or react to items affecting their system's operation.

Standards

The System Manager is in an excellent position, because of his intimate knowledge of the system, to develop and issue standards affecting system operation and user techniques. Management must ensure that this is done in light of business objectives and scrutinize this process, since the System Manager is also the "system police."

Some specific standards should include program segment size limits, (4K is recommended), and established maximums for code, stack and extra segments, to preserve on line response times for user systems. Compiles should be restricted to running only in batch mode or after hours, and privileged mode capabilities should be disallowed entirely. Along with the items discussed under application development which might be considered desirable standards, the System Manager should establish language standards as well. Some COBOL standards affecting the performance of the HP3000 systems include the following:

- o Data items should be described as picture "X" unless they are going to be used in some numeric calculation.....for example, the ZIP code should NOT be defined with PICTURE of 9's.
- o Numeric data items should be "signed" and defined with an odd number of digits and be "COMP" if less than 9 charact ers in length or "COMP-3" if greater than 9 characters in length.
- o Always "compare" and "move" fields of equal lengths.
- o Avoid using the "COMPUTE" verb since it treats all data items as packed-decimal and will convert the data to packed decimal before doing the computation and will then

restore the data to its original definition at the completion of the function.
- o Use indexing and NOT subscripting.
- o Put error messages in code, as literals, and not in data areas because on the HP3000 code is sharable while data areas are not.

Security

Systems security is the concern of everyone associated with the computer function. However, the responsibility usually falls to the System Manager who must ensure that the integrity of the system is not compromised and that the organization's data remains intact. The account structure and password protection of sensitive information should reflect the unique needs of each organization; but, in any case, must be a prime concern of the System Manager and must be continuously monitored. Dialup Lines provide one of the most vulnerable areas to a computer's security. If you have them, the System Manager must see to their security by allowing only authorized access. The lines should be "downed" at all times when not in use and only "upped" for known, authorized users; and only then with password protection.

Backup & Disaster Recovery

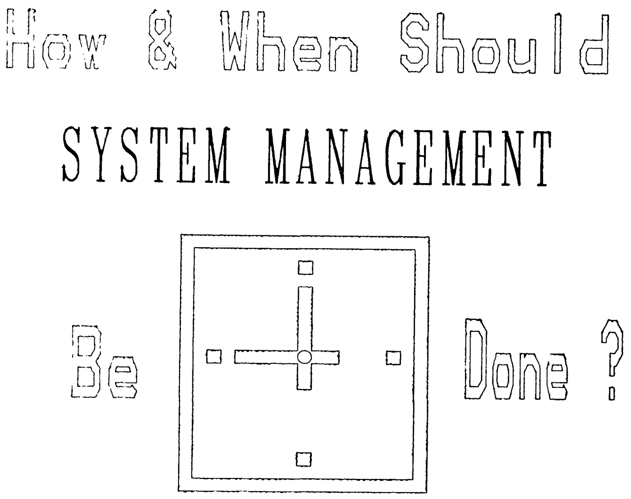
The best insurance a System Manager can subscribe to is a solid backup procedure which, unless required more frequently, should consist of nightly partials and weekly full system backups. To further support this insurance, it is recommended that a secure off site storage facility be provided for the various copies of these backups and that it be used religiously. At least three generations of backups should be rotated regularly. Monthly copies should be retained for at least six months to a year. Certain legal requirements might also dictate that some data be retained for several years.

The System Manager is the individual who is on the spot if for some reason disaster strikes. The System Manager must have a disaster recovery plan. The plan should be published, understood by everyone involved in effecting the recovery, and hopefully it will have been rehearsed. This is the one area most frequently neglected; but one has only to suffer a disaster once to realize just how very important a disaster recovery plan can be to the System Manager, their management and the business.

HP Liaison

Lastly, the System Manager should be your liaison with Hewlett Packard insofar as system matters are concerned. The System Manager should be officially designated to HP as your PICS caller and be the HP contact for maintenance, system upgrades, etc. By having the System Manager acting as the focal point for all HP support activity, you gain a certain degree of continuity which saves both time and resources when coordinating activities or in the handling of problems, since all of HP's field support organizations are designed to respond to all System Manager's requests. This includes not only PICS calls but on site assistance as well as account and problem management.

SLIDE #8



How & When Should

SYSTEM MANAGEMENT

Be Done ?

When & how should system management be done?

System management is not something that can be done on a hit or miss basis, when the moon is full and the boss is out of town. Gone are the days when the System Manager can judge system performance by the number of user phone calls received per hour. To be effective, system status should be looked at daily, just for drill, whether there are problems or not. Often performance patterns can be spotted long before they become major system problems.

Major housekeeping for systems performance should be done at least monthly so that hidden problems don't have too long to sneak up on you and become "serious."

As to the question of "How" system management should be done, I can only say, "carefully!" Actions which are not thought out thoroughly in this area can be much worse than the situation you are trying to resolve...Be very careful!

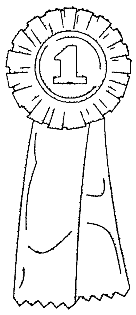
III. Conclusion

The benefits of having a designated and trained individual as System Manager are many and should not be taken lightly. When major systems problems occur, the trained System Manager is the only person who can effect an intelligent recovery from a system standpoint. The HP operation that today tries to get by without benefit of a person whose primary task is system management is doing just that, "getting by." That organization has given up on optimum system performance and is likely to be surprised at some point in time and will never be able to plan computer expansions or upgrades without jeopardizing their operation or making costly mistakes. They do not control their destiny but rather are at the mercy of the gods of the system.

SLIDE #9

The SYSTEM MANAGER and the function

- it is vital
- it must be done
- it is not difficult
- it is not free
- it does take time
- it is worth while



The System Manager and the function, therefore, is vital; it must be done; it is not difficult; it is not free; it does take time and it is worthwhile.

Mr. Tom Idema is Manager of Systems and Programming for the Furniture Systems Division of Westinghouse Electric Corporation. He is a graduate of Ole Miss, with an MBA from Western Michigan University. Tom is a member of the faculty of Grand Valley State College and Grand Rapids Junior College and has taught both management and data processing classes for the past nine years. He is both a Certified Systems Professional and Certified Data Educator. He has had several articles and papers published in national data processing journals and has presented papers at past INTEREX conferences and at various user group meetings. Tom is currently a member of the INTEREX Board of Directors.

3073. FITTING PRINTER TECHNLOGIES WITH PERSONAL COMPUTER AND OFFICE APPLICATIONS

Duane Schulz
Hewlett Packard Company
Vancouver, Washington

INTRODUCTION

A POSSIBLE SCENARIO

Suppose you are an MIS manager or an Office Systems support person, and you have finally gotten the interest of your company's Legal Department, who have been skeptical of your network's ability to provide adequate word processing and records management capabilities. They have extensive boilerplate word processing and list management applications. Not wanting to take any chances, you provide them with the most current hardware, HP150Cs, access to Personal Productivity Center, all of the finest HP150 software, and, knowing that laser technology is state-of-the-art, a LaserJet printer.

During your carefully planned implementation over the next few months, the PC and software training go quite well, but there a few snags related to the printer. For one thing, the printer doesn't handle some forms that the Legal Department uses: carbon sets, NCR paper, etc. which are required by government programs. The print resolution on the user's letterhead and bond paper, which they just reordered for the next year, is poor- it turns out their media is not consistent with the paper specifications for the printer. When the users discover the cost of consumables (toner, cleaning and filter cartridge, etc) and monthly maintenance, they are quite animated in their reaction. Service problems crop up - streaks appear on the paper, and HP asks why no one performs the suggested operator maintenance. After 7 service calls, you are told that your 175 pages per day usage exceeds the 2686's acceptable daily usage, and your service contract is in question. Finally, you find several application requirements which are simply not met for the user due to printer feature support constraints in the software they're using.

As a result, after four months, the manager of the Legal Department asks you to take your "solution" back - they love HPDESK, HPWORD and all of the other software and workstation capabilities, but if they can't get the final printed output with the quality they need (without a drain on productivity), it just won't work. And you end up with egg on your face.

ABSTRACT

Though this scenario may sound extreme, it happens frequently, and points out the importance of proper selection and placement of "workstation" printers in office and personal computer applications. A natural tendency is to blame the printer, but the true problems lie in MIS's knowledge of printer technologies in general, and methods of recommendation of your Legal Department's printers.

This paper will focus on the thought that, while MIS and User Support personnel are becoming quite knowledgeable regarding system and software capabilities, our knowledge of printer technologies and potential applications for these technologies lags far behind. We will provide some information in an attempt to improve this knowledge, and try to assist MIS and user support staff by providing tools for determining the best fit with applications and budgets. Our premise is that printer technology is as important as systems and software technology, and can be a major cause of application failure and system shutdown. Proper selection of printers can be the key to quality versus failure in office and PC applications. A printer's job is to generate images, and it is these images that people carry around with them when thinking of a department's capability to perform work.

To accomplish these goals, we will proceed with: an examination of why workstation printers are important; an overview of the printer industry and general issues confronting it; a review of information that MIS needs to select printers, and a method for making these choices; an evaluation of technologies used in workstation printers, including a comparison of specific attributes which owners should consider; and a look at some actions an MIS staff can take to gain control of this valuable portion of an integrated systems network.

THE IMPORTANCE OF WORKSTATION PRINTERS

To begin this examination of workstation printers, we should first outline just which printers are included in this category. For the purposes of this discussion, workstation printers are all printing devices which are: A) used specifically alongside individual user workstations; and B) are priced at $3500 or less (U.S.). HP printers in this category include the HP ThinkJet, LaserJet, 260X, 2932/4, 82906, and 267X. Other common offerings include devices produced by Epson, Olivetti, NEC, Diablo, Canon, and others. Though workstation printers are usually limited to low usage, they are commonly found in multi-user ("shared" or "workgroup") applications. Discreet technologies used to produce office and workstation printers which will be examined in this document include:

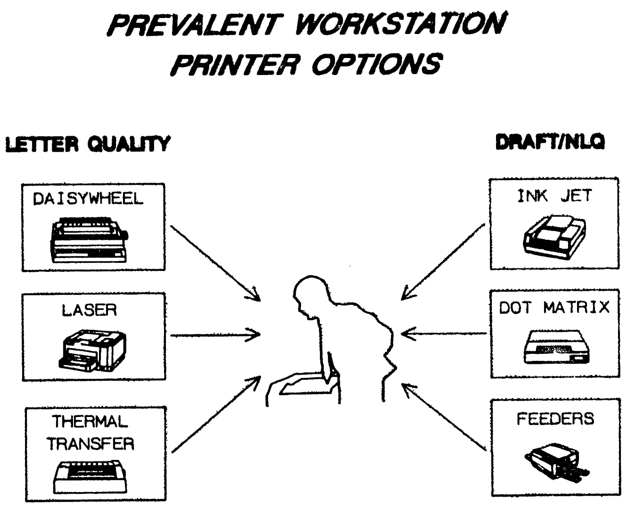


Figure 1 - Workstation Printer Options

There are several factors which contribute to the importance of these devices in an MIS environment (during this discussion, we will refer to MIS management, PC and Office Automation specialists, and User Support staff as "MIS staff," with "MIS" intending to describe a network of computers which is used together at a departmental - or larger - level):

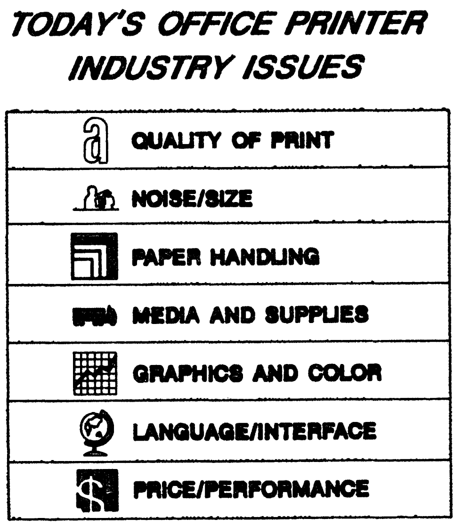
1: Workstation printer technology is moving at least as quickly as software and processor technology, making choice, fit and investment very difficult.

2: These devices will account for up to 50 percent of a typical workstation's cost; maintenance and ownership costs will be significantly higher than those for the workstation itself.

3: Printers are frequently the key to user acceptance of an application. Because they offer the image and result of an application, no application is truly complete without the appropriate workstation printers in place.

TODAY'S PRINTER INDUSTRY: ISSUES AND TRENDS

Before looking at how MIS can facilitate selection of a printer and comparing the available workstation printer technologies, it would be helpful to examine the printer industry in general, and to look at the issues and trends which are considered by successful workstation printer vendors. These include the items indicated below:



TODAY'S OFFICE PRINTER INDUSTRY ISSUES

- QUALITY OF PRINT
- NOISE/SIZE
- PAPER HANDLING
- MEDIA AND SUPPLIES
- GRAPHICS AND COLOR
- LANGUAGE/INTERFACE
- PRICE/PERFORMANCE

Figure 2: Today's Printer Industry Issues

QUALITY OF PRINT

A major issue for both printer users and vendors is quality of print. Since the advent of the electric typewriter and daisywheel printer, the standard of quality has been "letter quality," created by a single strike resulting in a fully-formed character, and offering potentially infinite resolution. Newer technologies have looked to the LQ character as a target for less expensive and higher-performance technologies. However, the dot matrix technology, created by striking groups of small wires, was adopted readily by data processing shops for applications which were not perceived to need LQ output. Continuation of dot matrix technology by using smaller impact wires and multiple passes across the paper have resulted in "near letter quality" (NLQ) dot matrix impact printers, also called "multi-mode" dot matrix printers. Finally, the advent of non-impact technologies, such as those used in laser, ink-jet, and thermal transfer printers now offer R&D facilities new ways of achieving letter quality printing.

SIZE/NOISE

As these technological advances have been made, other factors commonly discussed in input from customers are the size of the unit and the acoustic noise level. From the HP ThinkJet and other small printers to the HP2934, HP2601 and LaserJet, there is a considerable range in size now available from .3 cubic feet to over 2 cubic feet.

Given moves to standardize and improve the office environment, acoustic noise is frequently the major issue with office printers. Vendors now offer products which range in decibel output from "silent" to over 67 decibels of acoustic noise. This is a major area of research in many labs, since print quality often diminishes with noise output. A caution should be issued here, as many vendors use "specsmanship" to understate their noise level. Clearly, laser, ink jet, and thermal technologies are forcing this issue with more conventional impact printers.

PAPER HANDLING

This area is one of the most commonly-overlooked portions of the office printer market, and probably has the greatest impact upon the productivity improvements seen by the end user. Just a few years ago users were offered unidirectional forms tractors, manual paper feeding, or unreliable electronic paper feeders (usually with poor or limited application support). Today, graphics output and higher-volume office applications such as mailings have forced vendors to produce bidirectional forms tractors and simple and cost-effective sheet feeders, frequently with the ability to handle letterhead, plain bond, and envelopes concurrently. In fact, this topic is so important that a specialized industry in its own right has appeared in the form of several feeder manufacturers.

MEDIA AND SUPPLIES

The improved technology found in current workstation printers is usually associated with some costs, and these costs are otfen a surprise to new printer users. Variables such as print density, ink chemistry, feeders, duty cycle, impression-generating techniques, and paper tolerance have led to special media and/or consumable requirements. For instance, the HP ThinkJet requires special paper for best resolution, and uses removable ink cartridges which are printer-specific. Laser printers can require special paper, toner, ozone filters, fuser cleaning pads, OPC belts, and other consumable items. These are rarely included in a user's departmental budget, and rarely optional. Also, advances in paper production now offer new types of continuous forms, with laser-cut microperforations, removable backing and other capabilities. Again, the availability and cost of custom forms often comes as a surprise to users, and may never be communicated at all.

GRAPHICS AND COLOR

Without question, two of the most discussed topics in the printer community today are the ability to print high-resolution graphs, and the ability to do it in color. Since daisywheel and other fully-formed character printing uses a single strike per character cell, graphics is not a viable option. Sometimes users need graphics output more than letter quality output and will opt for dual-mode dot matrix printers because of their high print resolution (though many more seem to need letter quality print over graphics output). In reality, however, only high-density ink jet, laser, or thermal transfer technology will perform text and graphics output acceptable to selective office users, and these cannot handle special multi-part or odd-sized forms well. The need for integrated graphics in workstation printers will eventually spell the demise of the fully-formed character printer, but users who are ready and able to use this capability are rare, and that demise will probably not occur for years.

Equally difficult to resolve is the need for color output in workstation printers, which is available only on ink jet and dot matrix printers at this time. The issues at the moment revolve around quality of colors, resolution, speed, and software support. Again, this is a highly-emotional area, and will probably not be resolved until host software capabilities expand to offer examples of potential color applications to the user community.

COMMAND LANGUAGE AND INTERFACE

A frequently unknown technical issue relates to the command language to which a printer responds. This again offers users a surprise when software which performs a function (such as bolding or graphics output) does not properly ask a printer to perform this function. This is because several "command languages" exist for office printers, often with a separate set for each technology, and software packages are written to support only specific (not all) command languages. Common command languages for daisywheel printers include Diablo API2, Diablo HPR05, NEC, Qume 11, and Olivetti. Dot matrix printers use Epson, PCL (HP's Printer Control Language), and others. Lasers often offer a hybrid. Because software uses a specific command language or languages, it usualy offers inherent limitations to printer support. An effort is underway to adapt all HP-written and HP-provided software to HP PCL, which should improve printer support conditions considerably.

Also, consider that printer interfaces commonly found in HP networks include: HPIB, RS232, RS422, Centronics, and HPIL. Software designers must also attempt to insure that support of HP's 15-plus international character set layouts is provided by each printer's character generator, and so one sees that language (human and printer) and interface selection can be a confusing and often disappointing issue. Again, character set support is seeing an attempt at resolution through the emerging standards of IBM-8, ECMA-8, and, most importantly, HP's ROMAN-8, which offers an 8 bit character set which handles most major western languages.

PRICE/PERFORMANCE

Given all of these technological advances and challenges, it would seem that user and vendor alike are offered more than enough choices to make. However, manufacturers have found that newer manufacturing techniques as well as the inherent design of each tecnology are offering considerable opportunity to lower printer prices. In fact, there is presently so much development in the area of workstation printers that a sort of price war is in effect, with average prices dropping by as much as 20 percent each year. As an example, here are several comparisons of printer price and performance:

TECHNOLOGY	SPEED	RESOLUTION	1980 PRICE	1985 PRICE
Daisywheel	45 cps	Fully formed	$4,000	$1,500
Dot Matrix	200 cps	90 x 90	$4,000	$595-3,000
Ink Jet	150 cps	192 x 96	N/A	$495
Thermal Trans.	45 cps	200 x 200	N/A	$1,395
Laser	8+ ppm	300 x 300	$120,000	$3,495

Given this information and projection of high resolution color dot matrix printers in the $300 range and daisywheel printers in the $595 range, the future looks rather bright for the consumer. Unfortunately, the details of selecting a printer will offer severe limitations to the options MIS can offer users, and price can be VERY misleading. Below we will look at some important considerations which will determine which printers can be offered to users in the real world, and what price/performance factors are realistic.

STEPS TO PRINTER SELECTION

In order to recommend a printer to a workstation user, five basic items need to be considered. In many cases, only 1 or 2 of these variables are considered, and this leads to most printer problems and surprises. These five selection criteria are shown below:

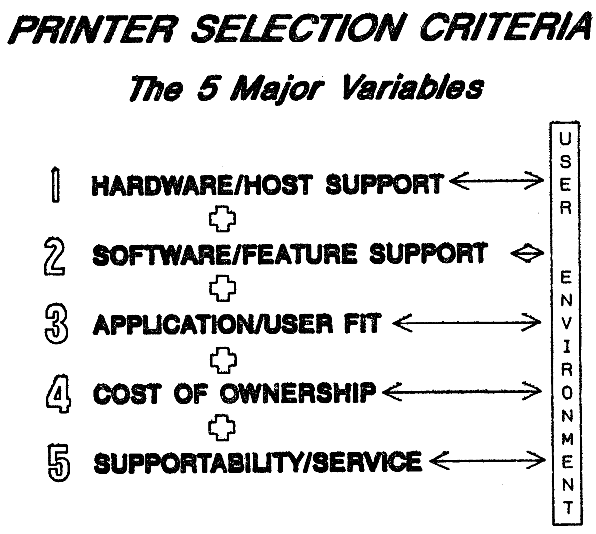


Figure 3: Printer Selection Considerations

HARDWARE CONSIDERATIONS

First, host support must be verified. This involves identifying the printer interface, cabling and data communication options. Once these are known, the host specifications should be checked against them for hardware supportability. For instance, an HPIB printer on an HP2628 workstation is only supported if the workstation is equipped with that option. Two types of support can usually be found: Explicit and Emulation support. With explicit support, the printer is fully supported by model number, such as the "HP2602" in the HP150's device configuration program. Emulation support involves using another printer's explicit support and your printer's emulation of that printer's attributes. This would include such things as using a "Diablo-compatible" daisywheel printer as an HP2601. The major risk associated with emulation support is that the vendor will usually not help with problems which may crop up, since they have not fully tested the other printer. Finally, other support parameters such as host speed limitations and printer spooling support should be identified and considered.

SOFTWARE CONSIDERATIONS

To determine software support, it will be important to know the printer's command language and print features, including: speeds, pitches, modes, graphics support, print enhancements, etc. Given these, each target software product to be used with the printer should be checked, again looking for explicit or emulation support. It is also worthwhile to determine support of the proposed printer with software which may be offered to the user in the future. Aside from specific print driver support of the printer, each printer feature identified should be checked through the software. For example, many printers offer normal, compressed and expanded print modes, and both bold and shadow print, but few word processing packages will support all of these combinations. Also, many printers (even daisywheels) offer graphics support, but many graphics packages, especially those found on HP3000 hosts, do not support these capabilities on office printers.

APPLICATION FIT

Though this consideration would seem to be obvious, it is frequently omitted from the selection process. Here, several less objective considerations, such as print quality, speed, acoustic noise, paper handling options, and size should be demonstrated to the user. Since the end user will be living with the device, it is crucial that they have an opportunity to accept how a printer performs in light of these variables. In many cases, no currently available (or supported) printer will meet user expectations, and demonstrating the features and limitations of each potential printer will help MIS by gaining empathy with our technical position.

COSTS

When a printer has been selected for placement in your MIS network, the total cost picture should be developed. This includes two costs: true purchase price and cost of ownership.

The true purchase price is simple to compute. It should include the price of the printer less any discount, plus the cost of attachments (ie. sheet feeder and/or tractor), cabling, a stand or table, and initial media and supplies, especially font cartridges and such items. Again, it is important not to omit these other items when considering total cost. Many users find that, when their printer arrives, it has no place to sit or no paper path, and when the first ribbon runs out, there are no more to be found.

The cost of ownership includes two variables: support costs and cost per page. The support cost should be an easily-obtainable monthly fee. If the printer is covered on a per-call basis, then you can use the annualized failure rate times the average cost of repair, then divide by 12 (these items should be available from your supplier). The cost per page is more complex, involving paper, ribbon, and other consumable costs. A formula for determining a printer's cost per page is shown below:

```
     COST PER PAGE =

          COST OF PAPER PER PAGE +

          COST OF RIBBON (OR INK CARTRIDGE) /
               (RIBBON-CARTRIDGE LIFE/AVG. CHARS. PER PAGE)
```

In the case of laser printers, the cost of ribbon or cartridge should be expanded to include all consumables, assuming the vendor's stated characters per consumable item (filters, toner, etc.)

As an example, let's look at a daisywheel printer with a 600,000 character ribbon at $19.00, and 25% cotton bond paper at a cost of 2.9 cents per page, also assuming 1500 characters per page (cover letters will average approximately 1000 characters or less per page). The cost per page for this printer would be:

$$.029 + (19.00/(600{,}000/1500)) = 7.65 \text{ cents per page}$$

Assuming an 80 page per day volume (22 work days per month) and $85 per month maintenance cost, the monthly cost of ownership for this printer would be:

(.0765 * 80 * 22) + 86 = $220 per month

Though these costs may often sound higher than we would like, the invoices will start rolling in after the printer starts being used, and it is helpful to insure that the cost of ownership is included in the user (or MIS) department's budget at the outset.

SUPPORTABILITY

Finally, it is also crucial to determine that the proposed application is consistent with the intended use of the printer. This is simple to ascertain. The printer's documentation should indicate usage (usually in pages or hours per day), environmental requirements, and other support-related parameters such as operator maintenance requirements. These items should simply be checked against the projected usage and intended user environment. This examination will eliminate any future suggestions that a printer is experiencing service problems because it is being used improperly.

Next, it would be worthwhile to examine some of the specific attributes of workstation printer technologies, and perform some comparative analyses of these technologies. This will be covered in the next section.

CURRENT PRINTER PRODUCT/TECHNOLOGY REVIEW

To offer a brief overview of the printer (and technology) options open to office and workstation users today, the following chart outlines the attributes and relative merits of each printer type. Followng this chart is a brief commentary on information contained within. To arrive at several of the numerical values on the chart, the following printers were used: Daisywheel: HP2601; High-resolution dot matrix: HP2934; Laser: HP 2686; Ink jet: HP2225; Thermal transfer: IBM Quietwriter. Also, please note that the dot matrix category refers to the newer high-resolution, dual-mode printers.

PRINTER TECHNOLOGY OVERVIEW

	DAISYWHEEL	HIGH RES. DOT MATRIX	LASER	INK JET	THERMAL TRANSFER
IMAGE CREATION	HAMMER	HAMMER	PHOTOELECT.	INK SPRAY	INK TRANS
1st MAJOR INTRODUCTION	1970	1978	1982	1982	1982
LIKELIHOOD OF ENHANCEMENT	LOW	HIGH	MEDIUM	HIGH	HIGH
POTENTIAL USERS					
Managers			X	X	
Professionals	X	X	X	X	X
Secretaries	X		X		X
Clerical	X	X		X	
POTENTIAL APPLICATIONS					
Word Processing+	1		2		3
Graphics		4	1	3	2
Spreadsheet	3	1	5	2	4
List Mgt.	1	4	3	5	2
Data Proc.	3	2	1		
INDUSTRY POSITION					
Noise	55-67 dbA	57-63 dbA	"SILENT"	<50 dbA	"SILENT"
Dot Resolution	FULLY FORMED	90 x 90	300 x 300	192 x 96	200 x 200
Size	3	4	5	1	3
Paper Handling*	M/T/F	T	F	T	M/T
Media Tolerance	EXCELLENT	EXCELLENT	GOOD	POOR	FAIR
Graphics	POOR	FAIR	EXCELLENT	GOOD	GOOD
Color Capable?	NO	YES	YES	YES	YES
PURCHASE PRICES	$1500-3500	$1200-2800	$3500-30000	$400-700	$300-1700
MAJOR REDUCTIONS LIKELY?	MEDIUM	HIGH	LOW	MEDIUM	MEDIUM
PRICE/PERFORMANCE**	$33	$14	$17.50	$3.30	$31
COST OF OWNERSHIP***	$241/MO	$84/MO	$279/MO	$36/MO	$224/MO

+ 1-BEST FIT, 5-WORST FIT ** PURCHASE PRICE/CHARS. PER SECOND

* M-MANUAL, T-TRACTOR, F-FEEDER *** 1500 CPP, FULL DUTY, W/MAINT., PAPER

Figure 4: Printer Comparison Chart

IMAGE PRODUCTION TECHNIQUE

The major differences in image fixation are non-impact versus impact, and surface versus absorbed images. The daisywheel printer still offers the only letter-quality impact technology, and the ink jet printer offers the only non-surface, non-impact image printing. These considerations are important for applications requiring lasting quality.

FIRST MAJOR INTRODUCTION/LIKELIHOOD OF NEW DEVELOPMENTS

Though most technologies see a "gestation period" of over 5 years in the lab, ink jet and thermal transfer technologies are the closest to the experimental stage, and high-resolution dot matrix printers can do more than the current market indicates, so these three printer types should see the most new product development.

POTENTIAL USERS

The best-fit printer of choice for each group of workstation users is: Managers: ink jet; Professionals: any; Secretaries: daisywheel or laser; Clerical: dot matrix, ink jet, or daisywheel. There are many potential exceptions and user preferences due to application needs.

POTENTIAL APPLICATIONS

The most likely printer for use with each major workstation application is: Word processing: daisywheel or laser; Graphics: laser or dot matrix; Spreadsheets: any; List management: daisywheel or laser; Data processing/report generation: laser or dot matrix.

INDUSTRY POSITION

Noise/Size:

The laser and thermal transfer printers are the only examples of "silent" printing available; however, the size of ink jet and narrow carriage dot matrix printers offer real advantages in space utilization and low heat generation.

Paper Handling:

This is another area where the daisywheel printer offers the most flexibility. Though dot matrix printers can easily adapt to feeders, they rarely offer the firmware or command set support needed to drive the triple bin feeder required in office applications. Also, note that manual feeding is difficult or clumsy with many printers, which can cause problems as well.

Media Tolerance:

Along with image-fixation techniques come limitations to the paper which the printer can tolerate. Most notably, all non-impact printers are limited in paper tolerance and flexibility when high print quality retention is expected.

Graphics:

Laser technology is the clear leader here, though ink jet and thermal transfer printers may soon offer resolutions nearer to the 300 x 300 dots per inch offered by laser printers.

Color Potential:

Through ribbon-lifting, multiple nozzles, or new photoelectronic processes, color capabilities CAN be added to all technologies but daisywheel. Nonetheless, vendors will be reticent to do this until high demand is seen - this is difficult and costly.

PURCHASE PRICES/LIKELIHOOD OF MAJOR REDUCTIONS

In this area, the notable conditions are the wide range of end user prices available. Many of the printers offered by major vendors are priced to reflect the cost of providing system, software and worldwide support and certification. Often, lower priced printers do not include these things, but their aggressive pricing will probably still affect major vendors' offerings in a slight downward trend over time. It is probably unrealistic to expect more than 10 percent price reduction per year in any area.

PRICE/PERFORMANCE

This measure of throughput per dollar shows that ink jet printing is clearly the best buy, with laser and dot matrix price/ performance still pretty competitive. Daisywheel and thermal transfer printers still command a premium for their adherence to true letter quality, which is the new Holy Grail in the industry.

COST OF OWNERSHIP

Including ribbon/ink cartridge/toner, appropriate paper, and service fees, and assuming usage at the printer's maximum monthly rating, these figures show the normally hidden costs associated with each technology. Again, the ink jet printer is the most reasonable to support, with the dot matrix still reasonably affordable. Laser, thermal, and daisywheel printers normally cost more to support due to the higher service costs and the higher cost of good cut sheet paper. These cost projections cannot be ovemphasized with the user.

CONCLUSION: AN MIS ACTION PLAN

Given all of these considerations regarding this important part of an MIS network, what can an MIS manager or user support staff do to prepare for coping with all of these choices? I would suggest six simple steps which can be carried out today, and these are described below.

USE AN APPLICATION APPROACH: For each request or problem, the determining factor for each printer should be: A) its ability to produce the final product of the user's applications in a fashion acceptable to the user, and B) the user's acceptance of the device in terms of human/operator interface, noise and size.

TRY UNITS BEFORE DEMAND IS SEEN: When an interesting technology or product appears, arrange to look at an evaluation unit, and become familiar with its capabilities, integration with hardware and software, and other attributes BEFORE you are involved in response to a request for user assistance in choosing a printer.

HAVE PRINT SAMPLES ON HAND: When discussing each type of printer internally or with user departments, do so while referring to print samples (the longer and more complex, the better). This will again keep eveyone focused upon the quality of the end result of the application. It is also interesting to see how print samples wear through constant use.

ADOPT SELECTION GUIDELINES: Using the "Steps To Printer Selection" above and/or other considerations, develop printer evaluation and selection criteria. Worksheets and review forms for MIS or O/A analysts are helpful here. This way, users can be made aware of the possible choices, and how you are willing to help them, before a need arises. This type of activity might even lead to an internal guide to workstation printers available within your network.

READ THE TRADE PRESS: Using office automation, personal computer, INTEREX, and data processing publications, review the newest hardware announcements (usually found towards the back of any issue), and get more information regarding printers which are of interest. Though many managers wait for their sales representative to announce a product, this is a good way to keep current.

MOVE WITH THE INDUSTRY: As with all electronic devices, workstation printer technology is moving quite rapidly, and it is important to be flexible in your ability to work with any technology which offers itself. Though diverse technologies are becoming available, the equally diverse users in your installation will react positively to different printers.

In this discussion, we have examined the issues confronting the workstation printer industry, looked at available printer technologies, and reviewed technical variables and selection techniques for MIS to consider. In putting this information into perspective, it is important to again consider the role a printer will play in a workstation application. Though "paperless" applications continue to be touted, when an accountant works with a spreadsheet, a secretary a report, a professional a market adoption chart, and a manager a business plan, these users will find, in the future, the PRINTED IMAGE which was used to prepare and preserve the information. Using the appropriate image for the task will help to gain their involvement in your MIS network as a daily part of their job.

BIBLIOGRAPHY

"Computer Users Catalog," Hewlett-Packard Company, Santa Clara, CA, December 1984.

Doub, James: "Personal Computer and Printer: A Team," Electronic Printer Industry Conference Proceedings, DATAQUEST, Inc., San Jose, CA, March 1984.

Downie, Robert: "Thermal Transfer Ribbons: The Gating Factor," Electronic Printer Industry Conference Proceedings, DATAQUEST, Inc., San Jose, CA, March 1984.

"Electronic Printer Industry Service," DATAQUEST, Inc., San Jose, CA, March 1985.

"New Technology Printers for Tomorrow's Office," DATEK Information Services, Newtonville, MA, August, 1983.

"Peripheral Support Guide," Hewlett-Packard Company, Boise, ID, March 1985.

"PRINTOUT Annual," DATEK Information Services, Newtonville, MA, February 1985.

"Vancouver Division Customer Engineer Handbook," Hewlett-Packard Company, Vancouver, WA, March 1985.

3074. THE ULTIMATE CHALLENGE IN APPLICATION DESIGN: MANAGING DATA INTEGRATION

Dr. Sreekaanth S. Isloor
275 Slater St., 10th Floor
OTTAWA, ONT., CANADA, K1P 5H9

ABSTRACT

An organization's investment in it's data is enormous. The enormity of this investment requires data to be treated as a corporate resource, even in cases where data is collected and managed by small subgroups within the total organization. The creation of this corporate resource depends on management commitment, user participation and standards. Achieving data integration in an environment where the development of numerous applications spans long periods of time is a formidable task. This paper outlines how data integration can be managed by the appropriate use of databases in such an environment.

1. INTRODUCTION

1.1 Value of Information as a Corporate Resource

In today's world of exploding information needs it is essential to manage information as a resource. The trend towards viewing data as a corporate resource is indistinguishable from the trend to manage information as a resource. Value is associated increasingly with the support data provides for effective business activities. If major strategic planning decisions are based on data, it is evident that bad and uncontrolled data will result in bad and inconsistent decisions. The value of data can be determined by the business cost associated with not providing accurate information whenever needed. When data is interpreted in terms of a business activity, it places tremendous responsibility on an organization. Assuming this responsibility requires control over all manual and automated data and all procedures used for the manipulation, communication and presentation of information in the course of doing business.

1.2 Objectives of Integrating Data

The software, the hardware, the firmware and the procedures that manage an integrated data base comprise a database management system (DBMS). A DBMS makes it possible to access integrated data across operational, functional, or organizational boundries within an interprise. The major objective of integrating data is:

a) to support management in strategic planning;

b) to facilitate a "one-stop shopping approach" through an information centre; and
c) to integrate all office functions [1] in an electronic office.

The benefits of integrating data are as follows:

* Different functions of an organization can be served effectiely by the same DBMS [2].
* Redundancy in stored data can be minimized, resulting in higher data integrity.
* Adequate security controls can be uniformly applied.
* Skilled personnel cost can be reduced by virtue of fewer people being required to develope, maintain and enhance application programs.
* Centralized control of the database is possible.
* Standards can be enforced on information managment procecess.
* Easier procedures for cumputer operations can be established.
* Phisical reorganization of the stored data is possible.

1.3 Long Term Implications of not Integrating

Data Redundancy:

If the same data is needed for several independent applications, it is often stored redundantly in several files. Such data redundancy requires multiple input, updating and reporting procedures. This results in numerous problems with the integrity of data.

Loss of Data Integrity:

Inadequate Data Integrity is a result of storing the same information in more than one place. Lack of data integrity can also result from poor validity checks on data. In a multi-user, multi-application environment, data redundancy demands synchronization of modifications to the database. It requires strict concurrency control techniques.

Data Availability Constraints:

When data is scattered in a number of data files, obtaining combined data of related different applications is somewhat constrained. In a fast moving business environment, availability of data to the right person at the right time is essential.

Difficulty in Management Control:

As a result of data redundancy and uncoordinated control, it is difficult to implement new guidelines or enforce standards across the organization.

Manpower Costs:

In a totally uncontrolled environment and in the absence of data integration, personnel costs for maintenance and development are exorbitant. James Martin in advocating a planned approach to control corporate data, states that "80% of the programming budget is being spent on maintaining or modifying past programs and data; only 20% is being used for new application programming". [3] This also results in duplication of effort.

The lack of data integration results in:

* a totally uncontrolled environment;
* misuse of software, human and computing resources;
* duplication of effort;
* loss of data integrity;
* inconsistent approach and lack of standards; and
* inability to identify the impact of change.

2. DATA INTEGRATION

The term integrated database refers to a database containing files from two or more applications which are not necessarily related. The term corporate database refers to the collection of all databases within an organization. Integration of systems means the amalgamation of normally unrelated applications to process some common data. This data may or may not be held in an integrated database. The benifits to an organization resulting from an organization resulting from an integrated database are as follows:

Availability of Infomation

Data stored on the integrated database by several independent applications is immediately available for query and reporting. Interrogation or reports on data from various applications can be accomplished by an end user oriented language. In effect any data from any application stored in the integrated database is available from that database directly.

Simplified Maintenance

All maintenance activities such as modification of definitions, addition of fields and changes in keys or utility programs are performed, controlled and logged by a central area.

Massaging/Sorting

In many applications there is a reduced need to massage and sort data for reports since the DBMS stores the data in logical sequence.

Standardized Techniques

Since all files are on the same database, naming conventions, coding conventions and documentation techniques are standardized. There will not be a requirement for multiple conventions and techniques. This should reduce any manual effort involved in coding, documenting and reviewing.

Centralized Support and Expertise

Detailed inhouse knowledge of the DBMS and its utilities will be centralized. There is no need for the programmers and analysts to be involved with the internals of the DBMS; those individuals can devote full time to applications and in meeting the end-user requirements.

Common Backup/Restore

Backup of the files for any or all applications can be scheduled and performed by a central area. Once backup requirements are established they can be invoked automatically. Since the database area is responsible for integrity of the database, it is also responsible for backup/restore of the database.

Restart/Recovery

Restart and recovery of any application is designed such that minimum effect on the integrated database occurs. Restart and recovery will be automated where posible to reduce the human intervention after system failure. These procedures will be performed by the centralized area and will require minimum involvement of the application area.

Utility Functions

Utility operations can be performed during online usage of the integrated database. The specific files requiring attention will be unavailable to that application during the certain operations, but all other files and applications are unaware of the utility.

Enhancements

Any enhancements for the DBMS announced by the vendor of interest will be installed and tested by the central area. The user will take advantage of the enhancement without requiring his involvement. All applications will benefit because of a modification to a common routine.

Space Utilization

Since the DBMS handles all space management there will be an overall reduction in the storage requirements for the integrated database compared to multiple single databases. Since the central area reviews space allocation regularly, and is monitoring performance of the DBMS and applications, corrective action can effect multiple areas immediately. There should be an overall reduction in running time/costs since applications are processing the intergrated files, instead of massaging and reformatting data from different files.

Other benefits of integration include reduced miscommunication, reduced duplication of effort and standardized procedures in times of failure.

Checklist provided in Appendix A can can be used to assess the actual integrated corporate data requirements in comparison to the present DBMS enviroment.

3. APPROACHES TO DATA INTEGRATION

While it is quite easy to say we will move to a database environment and maximize the integration of systems, there is phenomenal effort required to get there. Very briefly, the organization must:

* identify what it does and what data is used in doing it;
* determine the generic groups of data used in doing its business;
* establish funding mechanisms to support this philosophy;and
* assess the impact on people associated with changing the way things are done.

There are several strategies possible for attaining data integration in an organization. In general, there is no one strategy to be used by all. The particular strategy adopted by an organization is dependent on several factors, for example, the current environment, the degree of sophistication of applications, the role of the DBA [4], the spatial distribution, and hardware/software distribution of users, the level of standardization in data and methods and the degree of autonomy of the various users. Here the two extremes are outlined. In Approach A, applications are developed independently and subsequent data analysis and integration are attempted. In Approach B, the data analysis, data modelling and data administration are implemented first and then systems are developed based on common data.

3.1 Approach A

In approach A, each application is independently analyzed, designed and implemented. Upon successful implementation of all systems, an effort is made to integrate the common data. The approach involves:

Step A1: Analyze Data for single application.

Step A2: Design the database for the application under consideration.

Step A3: Implement the application.

Step A4: The above steps are independently applied to each and every application.

Step A5: Analyze all implemented systems to determine commonality of data for integration.

* Determine common data in all related applications
* Determine common data in unrelated applications
* Identify data in Common data that is standardized (e.g. Standardized codes etc.)
* Develope standards for data in common data that is not standardized.

justify on
Step A6: Design integrated database for common data based on analysis in Step A5.

3.2 Approach B

In Approach B, a conceptual global analysis of all the relevant systems and potential applications is carried out first to determine common integratable data. The applications are designed with the integratable data as a common base. To do this complete data analysis and data modelling is performed. The Approach involves:

Step B1: Conceptually, model corporate data. Collect information at Corporate level on entitles, and relationships between entitles.

Note: A little reflection about the data dictionary approach is that it tells you what information to collect and how to use it. The data dictionary provides Coordination and stability to the efforts it supports because it gives permanence to the information and specifications on which those efforts are based.

Step B2: Analyze existing application to collect minimum possible common data, by modelling entities and their relationships.

Step B3: Analyze future potetial applications by collecting information on potencial usage of data. Use results from Step B2 and above to determine a maximal set of common data.

Step B4: Design standard codes and standards for the maximal set of common data.

Step B5: Design the database for the common data.

Step B6: Design and implement all applications based on a common base of integrated data.

Remarks: Approach B, is an ideal procedure, whereas Approach A is at the other end of the spectrum. Approach B has an inherent advantage in stressing that the idea of standardizing data is a good one. Approach A is the result of lack of management commitment, lack of user participation and the inability of data processing departments to manage the Corporate data resources.

In the next section, based on a review of some hypothetical systems and their component data, a method is developed to allow progressive intergration. The common data is identified and integrated increasingly to provide short term benefits while preparing for the longer term. While the move to data integration is technically practical, and economically justifiable the major problems will be social.

4. EXAMPLE OF DATA INTEGRATION

A methodical approach to data integration is outlined in this section based on an analysis of some hypothetical systems. Let us consider three application systems SYSTEM 1, SYSTEM 2, SYSTEM 3 that are developed independently, but use considerable amount of data in common. It is obvious that, to handle inter system requests, a customized bridging system would become essential for shorter term. Let us call that system, SYSTEM 0. Let us assume that the generic groupings of data that are used in doing business with the help of above hypothetical systems are:

* Client Company data;
* Project Data;
* Financial Data;
* Commonly used coded tables data.

4.1 Common Data

Client Company Data

The systems considered have recorded data about a company in different business dealings. Therefore, the information about a company is stored under multiple systems. In this example, the company data should be stored as common data accessible to any system that needs it. The analysis of company data shows at least the following data elements common to some of the systems.

Client Company Data

Data Elements	Systems			
	1	2	3	0
Company Code	X	X	X	X
Company Name	X	X	X	X
Company Address	X	X	X	X
Zip Code	X	X	X	X
Telephone Number	X	X	X	X
Telex Code	X	X	X	X
Contact Name	X		X	
Year Established Date	X		X	
Mailing Address	X	X	X	X
Mailing Zip Code	X	X	X	X
Number of Employees	X		X	
Total Sales	X		X	
Stock Exchange Status		X		X

Similarly, an assessment of commonality of the project data, financial data and the coded tables data is given below.

Project Data

The project data common to the systems is a direct result of the commonality of business dealings of the companies involved. Thus, common data about projects is stored under multiple systems.

Data Elements	SYSTEMS			
	1	2	3	0
Project ID	X	X	X	X
Project Description	X	X	X	X
Project Value	X	X	X	X
Project Effective Date	X	X	X	X
Project End Date	X	X	X	X
Program Name	X	X		X
Sub-Program Name		X		X
Sub-Program Code		X		X
Activity Status Data	X	X	X	X
Program Code		X		
Project Budget Revisions	X	X	X	X

Financial Data

Financial data (e.g. Expenditure information) should be entered only once. ntegration of financial data will avoid considerable number of duplication of data entry operations, and lack of financial data integrity by avoiding redundancy. The following financial data is a representative set of common data in the systems under consideration.

Catagory of Data Elements	SYSTEMS 1	2	3	0
Commitments Information	X	X		X
Contract Value	X	X	X	
Project Expenditure Information	X	X	X	
Project Recovery Information	X	X		
Project Losses Information		X	X	X
Project Losses Recovered Info		X		X
Fees Received		X		X
Undisbursed Insurance		X		X
Outstanding Insurance		X		X
Project Amount Requested	X	X	X	
Revised Budget	X	X	X	

Coded Table of Tables Data

Coded Table of tables basically relates data names with their codes assigned by coding systems. Coding of same information in different systems must be standardized for data integration. Information on regions, for instance, is used in several systems.

Obviously, it is useful to store such information, in standard formats centrally, so as to facilitate its shared use by other system. A representative set of such information is provided below.

Entity Catagory	SYSTEMS 1	2	3	0
Country Information	X		X	
Region Information	X		X	X
Product Information	X		X	X
Responsibility Centre Information	X	X		
Product Support Information	X		X	X
District Information	X	X		X

4.2 Cursory Analysis of the Hypothetical Systems

In section 4.1, a brief analysis of data in the systems showed that there is considerable amount of common data. In this section an analysis of the effects of data integration with common data as a base will be considered.

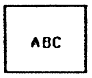
To begin with, represent data bases, systems and interfaces between systems as follows

4.2 Cursory Analysis of the Hypothetical Systems

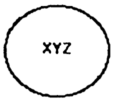
In Section 4.1, a brief analysis of data in the systems showed that there is considerable amount of common data. In this section an analysis of the effects of data integration with common data as a base will be considered.

To begin with, we shall represent data bases, systems and interfaces between systems as follows

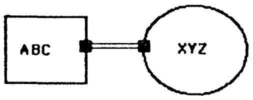
Legend



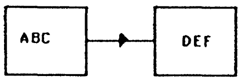
= ABC is a system such as SYSTEM 1, SYSTEM 3



= XYZ is a set of common data such as CLIENT COMPANY DATA, CODED TABLES DATA.



= ABC is a system that uses common data XYZ.



= ABC and DEF are systems with an interface that transfers some data from ABC to DEF. The interface may be MANUAL or AUTOMATIC

Configuration A

The present situation in most organizations without any integrated data can be represented as follows. Each system maintains and controls its own data. The common data is stored and maintained separately by every system.

SYSTEM 0 which acts as a support system for systems such as SYSTEM 1, SYSTEM 2, SYSTEM 3, etc. interfaces to every system to supply some financial data through either manual or automatic interfaces. SYSTEM 4 does not interface to any system. A representative set of problems encountered in this configuration is listed below due to the lack of data integration.

Configuration A

The present situation in most organizations without any integrated data can be represented as follows. Each system maintains and controls its own data. The common data is stored and maintained separately by every system.

SYSTEM 0 which acts as a support system for systems such as SYSTEM 1, SYSTEM 2, SYSTEM 3, etc. interfaces to every system to supply some financial data through either manual or automatic interfaces. SYSTEM 4 does not interface to any system. A representative set of problems encountered in this configuration is listed below due to the lack of data integration.

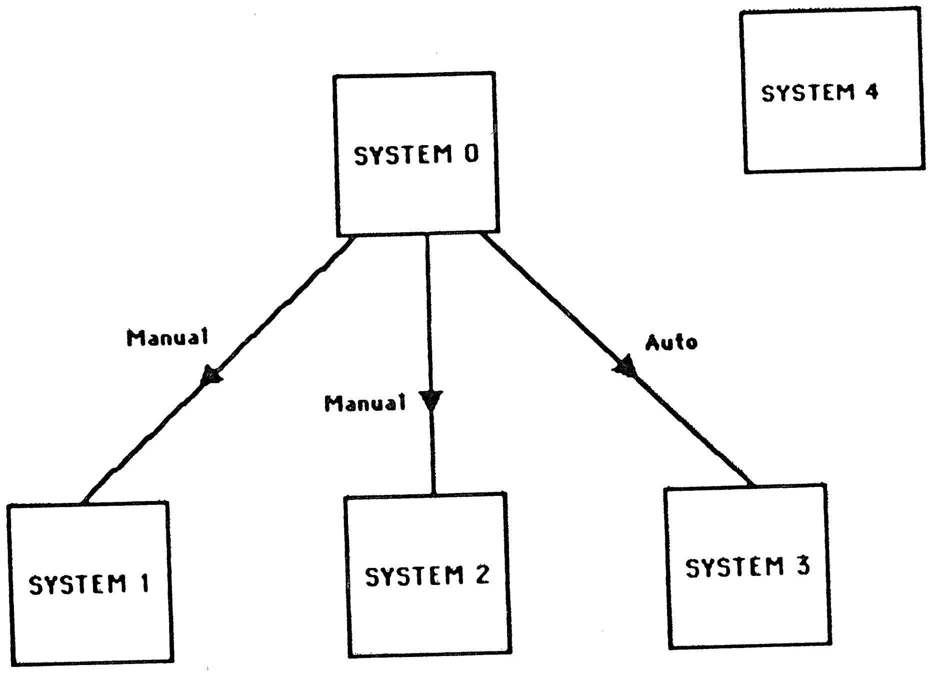


DIAGRAM : CONFIGURATION A

Problem 1: A SYSTEM 3 project recorded in SYSTEM 0 but not recorded in SYSTEM 3 is a result of different project identifiers for the same project. This can happen due to (i) SYSTEM 3 forgot to record the same project, (ii) SYSTEM 3 used the wrong identifier, and (iii) SYSTEM 0 used the wrong identifier. In this situation, determining whether SYSTEM 3 or SYSTEM 0 is to be corrected is not simple.

Problem 2: There is no guaranteed way to check if a single client company has defaulted. For instance, a company "A" blacklisted by SYSTEM 3 for not satisfying contractual obligations may do business with the group responsible for SYSTEM 2, even though a tentative organization decision may be never to consider company "A" again.

Problem 3: As regards financial data, there may not exist one consistent definition for "committment value" in different systems. Each system may have its own definition (explicit or implied) of "committed value". This is a stumbling block for budgetary control. For instance, reconciling a SYSTEM 1 committment with a SYSTEM 0 committment may pose problems.

Configuration B

In this configuration, the basic common data about companies and projects is shared by most systems. This configuration would be a natural successor to the configuration A. Rest of the data such as financial is still maintained separately by each system.

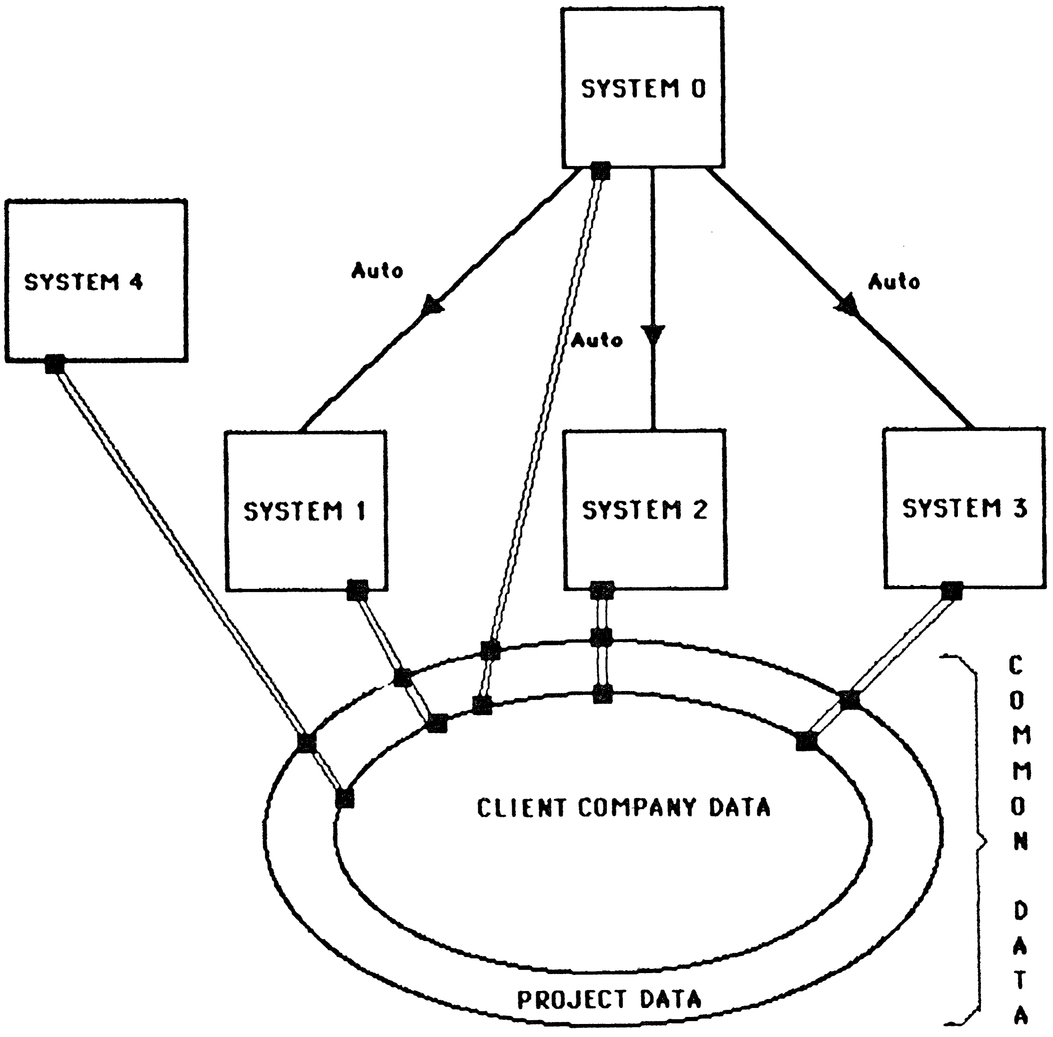


DIAGRAM : CONFIGURATION B

Configuration C

In this configuration, the common data about companies, coded tables, projects is shared by all systems.
The financial data is stored as common data on all projects through a system called here the "financial system" (FINSYS). FINSYS generates, maintains and controls all the financial data that is required for every system. Please note that SYSTEM 0 as a supporting, bridging system is not necessary.

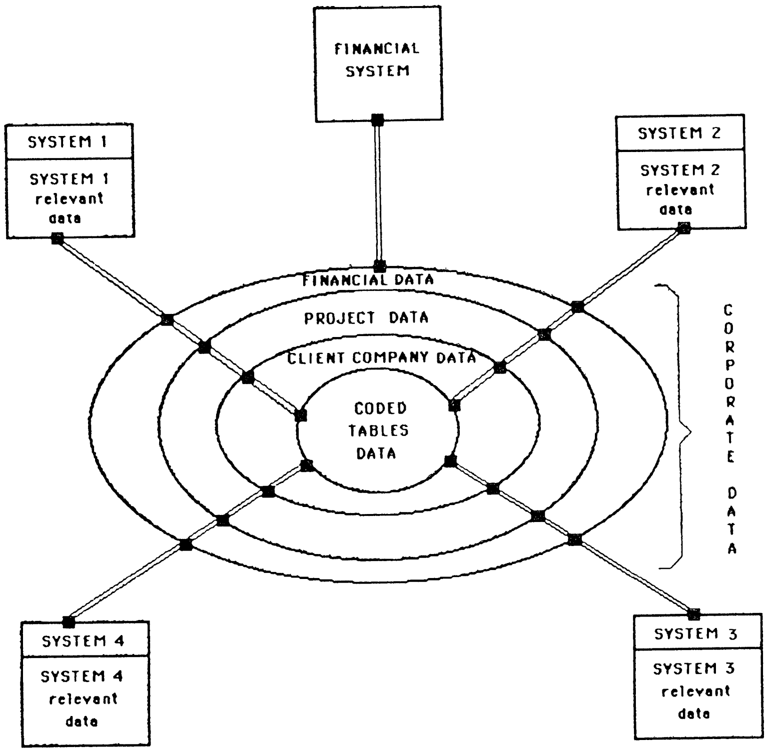


DIAGRAM : CONFIGURATION C

Remarks

a) Configuration C is closer to an ideal integrated data base system. Configuration C would be the result of using Approach B described in section 3.2 to integrate data bases.

b) Common standardized project numbers, codes in coded table of tables data, standardized company identifiers will eliminate all the problems that were encountered in configuration A.

c) Controlling financial data through a centralized system such as FINSYS (hypothetical) described in Configuration C results in better financial data security and integrity. It also alleviates problems such as problem 3 described in configuration A. Accessibility of financial data for budjeting and planning purposes at the organizational level enhances considerably with with centralized system control to the data.

4.3 Access Control for Common Integrated Data

The data items for common integrated data should be centrally agreed upon. Centralized control is needed to ensure that the sharing of common data between applications will be possible. A data dictionary should be used by all teams that are developing applications using common data. Only with proper control on common data, it is possible to avoid the crippling problems that result from piecemeal development. A delicate balance between central control of common data and decentralized development of applications is needed.

In an integrated database such as shown in configuration C, update control should specifically be assigned with one application or some central authority. For instance the financial data updates should be controlled only by FINSYS (the hypothetical system). Individual systems such as SYSTEM 1, SYSTEM 2, SYSTEM 3 should not be allowed to control any financial data. Similarly the responsibility for update control of shareable Client Company information should be assigned as well. The data on coded table of tables is used by most of the systems.

A rough analysis of the update control necessities can be performed by using the checklist in Appendix B to determine a fair, acceptable assignment for controlling data updates.

5. CONCLUSIONS

In summarizing, an organization must consider a global analysis of its data to plan an effective data integration strategy. However, to achieve configuration C of integrated data from the present configuration A, it is necessary to select a data integration Approach that falls between two suggested extremes in section 3. Configuration C is achievable from A via configuration B by choosing suitably planned strategy.

REFERENCES

1) S.S. Isloor, "4th Generation DBMS as the Heart of the Systems of the Automated Office", Proc. HP3000 IUG Conference, Amsterdam, April 1985

2) E. Miklovich and S.S. Isloor, "Selecting the Right DBMS for Your Environment", Proc. CIPS Session "82, Saskatoon, Canada, May 1982, pp. 87-94

3) J. Martin, "principles of Data-Base Management, Prentice-Hall, Englewood Cliffs, New Jersey, 1976

4) H. M. Weiss, " Making the DBA the key to System Development", Canadian Data Systems, March 1981, pp. 66-70.

APPENDIX A

CHECKLIST FOR INTEGRATED CORPORATE DATA REQUIREMENTS

The following is a checklist that is representative and not necessarily exhaustive for determing integrated data requirements.

1. Does the integrated data satisfy today's needs for information in multiple applications and of the information centre?

2. Does it satisfy today's needs within a reasonable time to match performance requirements?

3. Does it (to a large extent) satisfy all the anticipated, and (to some extent) unanticipated requirements of end users?

4. Is the integrated corporate data easily expandable to keep pace with the expansion of the enterprise or organization?

5. Is it easily modifiable in a changing hardware and software environment?

6. Are good mechanisms in force to maintain the correctness of integrated data over the duration of usage?

7. Are validation checks in force to ensure that consistent new data is entered into the data base?

8. Are access control and authorization techniques adequate to control access to stored data?

APPENDIX B

CHECKLIST FOR UPDATE CONTROL REQUIREMENTS OF COMMON DATA

A representative checklist for update control is provided here.

1. Is the database obtained from multiple sources?

2. Does the data need to be updated from multiple locations.

3. Is the data used by multiple applications systems?

4. Which application system has statistically most used the data?

5. Which application system has statistically most updated the data?

6. Can the data be updated overnight in batch?

7. In case batch updates are performed, is the data sufficiently up-to-date for most users query traffic?

8. Which application system has the best data entry controls?

3075. Natural database normalizing: The Entity-Relationship Approach of IMAGE/3000.

F. Alfredo Rego

Adager
Apartado 248
Antigua
Guatemala

The IMAGE/3000 database management system has unique mathematical properties which are natural consequences of its original design criteria. These IMAGE properties allow us to model our entities and their relationships in a normalized fashion, without the convolutions required by run-of-the-mill database management systems described in the literature of the day.

In this essay I describe some of IMAGE's most intriguing mathematical properties and I give specific examples of their applicability to common database challenges.

From the call for papers

The call for papers from the Baltimore/Washington HP3000 RUG has a paragraph which highlights the general database challenges:

> "The conference theme -- INFORMATION CROSSROADS OF THE 80's -- reflects the enormous volumes of data that the hardware and software of today are capable of collecting and the equally staggering problem of turning that raw data into meaningful information".

Let's review some of the fundamental issues and some of the specific solutions, particularly those related to the HP3000 computer and its database management system: IMAGE/3000.

Database buzzwords

We use databases to model the behavior of ENTITIES and RELATIONSHIPS.

A DATA ENTRY represents the ATTRIBUTES of an entity or a relationship.

A KEY is an attribute (or a COMBINATION of attributes) which uniquely identifies a data entry.

A DATABASE is a collection of DATA ENTRIES.

A Database Management System (DBMS) attempts to control things. For EFFICIENCY's sake, a DBMS has some type of internal structure to FIND and ASSEMBLE data entries. For CONVENIENCE's sake, a DBMS has some type of USER INTERFACE to CREATE, MAINTAIN, and RELATE data entries to assemble INFORMATION.

The data entries which the user "sees" through the interface may be REAL (if they exist physically in the database) or VIRTUAL (if they are the result of relational operations on real or virtual data entries).

INTERFACE vis-a-vis STRUCTURE

The USER INTERFACE is an "ambassador" between the raw bits-and-bytes computer stuff and the human-like specifications of the end-user.

A LOUSY user interface imposes the restrictions of the STRUCTURE upon the poor suffering user. A NICE user interface shields the user from the structure's shenanigans, while still being able to take full advantage of the structure's properties. A good user interface is as efficient as possible without being obnoxious.

An interface knows the "internals" of the database structures as well as the "externals" of the user desires, and spends its existence translating back and forth between bits and thoughts. This may very well be the fastest kind of shuttle diplomacy!

Online database applications

Online database applications should provide RELEVANT information while somebody waits over the counter or over the telephone.

This means that we should design (and periodically tune) our database systems to provide the fastest possible response time for the most important transactions and queries. Ideally, in a simple way!

Complexity and NORMALIZATION

Unfortunately, things ARE complex. But we should avoid UNNECESSARY complexity. This is the objective of NORMALIZATION.

The issue is to place the appropriate resources (no more and no less) where they belong, at the appropriate level, at the appropriate place, at the appropriate time. And to be able to reallocate resources quickly and effectively to "balance the load".

This means that normalization applies at every level in the global computer hierarchy:

- Data entry.
- Dataset.
- Database.
- Computer.
- Node.
- Network

A normalized network is open-ended. We can add more elements at any layer without affecting existing systems. We can delete elements from any layer without affecting existing systems which do NOT access such elements.

This, in a nutshell, is the challenge of normalization:

Do we want to favor efficiency in terms of ACCESS or do we want to favor efficiency in terms of MAINTENANCE? In general, the higher the normalization (i.e., the scattering), the higher the communications and coordination costs.

Normalization is neither good nor bad. It is simply a method which allows us the freedom to choose HOW FAR we want to go in one direction or the other in the NORMALIZATION SPECTRUM, which has HIGHLY UNNORMALIZED DATABASES at one end and HIGHLY NORMALIZED DATABASES at the other.

Usually, efficiency in terms of access implies redundancy. But redundancy, in itself, is not "bad". It is just more difficult to maintain a bunch of redundant things in perfect synchronization. This is analogous to an "orchestra-man" who must play all kinds of disparate instruments in a (more or less) coordinated fashion. (Don't try to rearrange a particular piece of music 5 minutes before the next concert! All hell will break loose!)

Usually, efficiency in terms of maintenance implies simplicity of roles AND a multiplicity of role-players. If we want to change ONE role, we only have to change one player. But it can be a drag to keep track of thousands of players. This is analogous to those fascinating groups of musicians who play bells, one bell per musician. Each person is a "specialist" who can only play one note.

In terms of maintenance, we can see how difficult it would be to "tune" all the instruments of the orchestra-man while he is performing and how easy it would be to simply exchange a bell which is out of tune in the case of the multiple-musician single-note per person group. Nobody would get thrown out of balance!

A super-normalized database contains a variety of small data entries, grouped together in a multiplicity of datasets, with many instances of key fields scattered all over the place. Even simple queries may require that we assemble the information from many sources. But we have a better chance that each of these sources is correct. It is simpler to maintain a "specialist" source up to date than to maintain a complex source which tries to keep track of everything at the same time (like our friendly orchestra-man).

Even though it may seem paradoxical, our experience shows that highly-normalized databases actually occupy less total disc space than unnormalized databases. Particularly if the redundant attributes tend to be larger than the keys. Which is usually the case: the name of an entity tends to be longer than its Number-ID.

This scattering of information into small bits and pieces might become a nightmare to control. Fortunately, IMAGE's path mechanism automatically protects the referential integrity of highly-normalized databases, since it requires that every detail key field MUST have a corresponding ChainHead in a master.

Naturally, we can go to ridiculous extremes and normalize a database to death. We could conceivably chop up the information about an employee in many data entries, each containing a single attribute such as name, birth date, salary, and so on. But this would really be splitting hairs! Common sense should prevent us from doing such attrocities. This is the motivation behind the rules for the fifth normal form: A data entry is in fifth normal form when there is nothing significant left to normalize!

Some general observations

In terms of "SPACE", an entity may be related to zero, one, or more entities (of its own class or of different classes).

In terms of "TIME", these relationships may happen all at once or they may happen one after another, in a strictly sequential fashion.

A relationship is an entity. For example, a marriage is a relationship between two people AND a marriage is also the subject of attention of a marriage counselor who treats it as an entity.

An entity is a relationship. For example, an individual is an entity AND an individual is a relationship formed by internal organs, genes, environment, and so on.

It is a matter of CONVENIENCE to designate some "thing" as an entity or as a relationship.

Usually, a relationship's key is a concatenated key, consisting of the keys of the related entities. If the same entities can be related in different ways (thereby giving rise to several data entries to represent the different relationships), then each relationship's key will include some additional attribute(s) which will differentiate this relationship from other relationships involving the same entities. (Example: line-numbers in a purchase order "PO-line" relationship which links customer, product, PO#, etc.)

Another example that shows the convenience of a concatenated key has to do with "discretionary" pricing (or discriminatory, or whatever you may want to call it). In this case, the price of a product for a customer may depend on the part's supplier, on the customer's rating within the company, on the order-date and/or on the ship- date, and so on. In other words, the price is an attribute of the RELATIONSHIP among all these entities; the price is not an attribute of the product alone.

Access strategies

Some people have spent endless amounts of time and talent solving a fascinating problem: How to minimize the effort required to answer the most infrequently-asked (and most arcane) questions.

Some other people have spent endless amounts of time and talent solving another fascinating problem: How to minimize the effort required to answer the most FREQUENTLY-asked (arcane or not) questions, while still preserving a reasonably efficient environment for those who must toil, on a daily basis, with the thankless task of feeding and babysitting the database.

IMAGE allows us the freedom to go "explorer-like" with sequential and direct access methods, which allow us to implement our own indexing methods, as well as "big-hub-to-big-hub-like" with hashing and paths. We do not HAVE to scan anything in a predetermined order. But it is nice to know that we may, if we know that a given "routine-route" (get the connection?) will get us to our desired destination.

In an online database system, we want to get information about given entities and their relationships. This means that we want to find the entry (or group of entries) of interest to us, among millions of entries, as efficiently as possible.

IMAGE provides two access methods which are optimized for efficiency: HASHING and CHAINING. In general, we access entities (in master datasets) by means of hashing and we access relationships (in detail datasets) by means of chains which we built as we added new detail data entries. These are contents-oriented methods.

We can always access either entities or relationships in other address-oriented IMAGE-access modes: serial or directed. As a matter of fact, this is what some indexing systems do, even though they are subject to chaos whenever secondaries migrate in master datasets or detail chains lose their absolute positions as a consequence of repacking.

Advanced indexing systems avoid address-oriented IMAGE-access methods and cleverly build upon IMAGE's intrinsic access methods (hashing and chaining).

Naturally, we may have valid reasons (usually having to do with convenience, performance, or both) that motivate us to use our own combinations of physical MASTER and DETAIL datasets, with or without physical paths, to model a given collection of entities and/or relationships. Usually, these valid reasons are dictated by our choices of indexing techniques.

A practical methodology based on DATA ENTRIES

A data entry models an entity or a relationship. A data entry has:

- A unique identifier ("KEY") for the represented entity or relationship. Any unique identifier can serve as a key. But some identifiers are more convenient than others.

- ATTRIBUTES (if any) which further "qualify" the entity or relationship.

The functional dependencies among keys and attributes will tend to show a remarkable stability, particularly if you cluster things around OBVIOUS (to you) entities and relationships. For instance, the functional dependency between a Personal Identification Number and the name of a person will probably hold for life.

The manners in which people access, combine, manipulate, present, and otherwise massage the data contained in the database to produce information (or misinformation) will tend to change according to the inevitable changes in the political winds.

Given these facts of life, it might make more sense to spend our limited energies and resources analyzing ENTITIES and RELATIONSHIPS first. Interestingly, we may find that this Entity-Relationship Approach will automatically and conveniently point us in the direction of a very desirable thing: We will find that we need very simple user interfaces in order to maintain information in the database and (most importantly, of course) to obtain information from the database.

Naturally, stability should not imply inflexibility. The challenge is to be as stable as possible while still being flexible and adaptable to changing environmental conditions. But there should be some back-bone to the whole thing!

Database Dynamics

Entities and relationships don't just sit there, frozen in time. They interact with one another and with their environments. They have trajectories through time and through space, analogous to the performance of a musical score or to the full-blown production of a ballet.

Events happen which affect (and are affected by) our entities and our relationships. This, in effect, is the essence of "being", as mentioned by Plato, who said that "Being is the ability to affect and be affected".

The discipline of DATABASE DYNAMICS studies the TRANSACTIONS which affect (and are affected by) the evolution of databases through time. Both in terms of the changing structures of the databases themselves and of the changing meanings and values of the various kinds of information contained in the databases.

We use PICOSECONDS (trillionths of a second) to measure events which we think are super-fast. We use AEONS (billions of years) to measure events which we think are super-slow.

Somewhere in the middle of this wide spectrum we find the events which occupy most of our attention in our daily concerns. By definition, these are the events which are the most useful and interesting. Most IMAGE databases, for instance, keep track of entities and relationships whose event-speed ranges from a "fast" which we can measure in days to a "slow" which we can measure in years.

Some observations on the IMAGE/3000 implementation

Since entities and relationships are "topologically equivalent", IMAGE/3000 uses the same construct ("DATA ENTRY") to represent either an entity or a relationship. For convenience (and performance) you may want to use MASTER datasets as repositories of entities and DETAIL datasets as repositories of relationships. After all, the designers of IMAGE tried very hard to help you out and they spent many sleepless nights optimizing these two kinds of datasets. But you can always change your mind to suit YOUR convenience!

Entities and relationships have identifying names or tags ("KEYS") and qualifiers ("ATTRIBUTES"). Please note that a key is a field (or a collection of fields) which uniquely identifies a data entry.

A key does NOT have to be an IMAGE search field. IMAGE search fields are defined only for performance's sake, to allow PATHS between master and detail datasets. Paths are particularly attractive for online access to fashionable relationships, since paths tell IMAGE to build appropriate linkages when adding a new data entry.

IMAGE/3000 manipulates entities and relationships with the same operations: addition (DBPUT), finding (DBFIND, DBGET), deleting (DBDELETE), modifying

(DBUPDATE), coordinating INTERFACE traffic (DBLOCK, DBUNLOCK, DBBEGIN, DBEND), etc.

The IMAGE user interfaces use these fundamental IMAGE/3000 operations to implement the standard user-interface bag of tricks (joins, projections, selections, averages, consolidations, relations, etc.)

The order of keys and/or attributes is arbitrary. Therefore, the sequence of fields ("columns" or "attributes") in an IMAGE/3000 data entry ("tuple", "row", or "record") is arbitrary. However, IMAGE provides the LIST construct to map ANY permutation of key(s) and/or attribute(s) from/to the user's buffers.

Remember that the database should remain as stable as possible even as the fads and fashions of applications designers shifts around. The LIST construct in IMAGE is one of the most powerful database tools in existence: It promotes database stability at the same time that it allows all kinds of whimsical permutations in terms of sequential presentation (to and from screens, for instance).

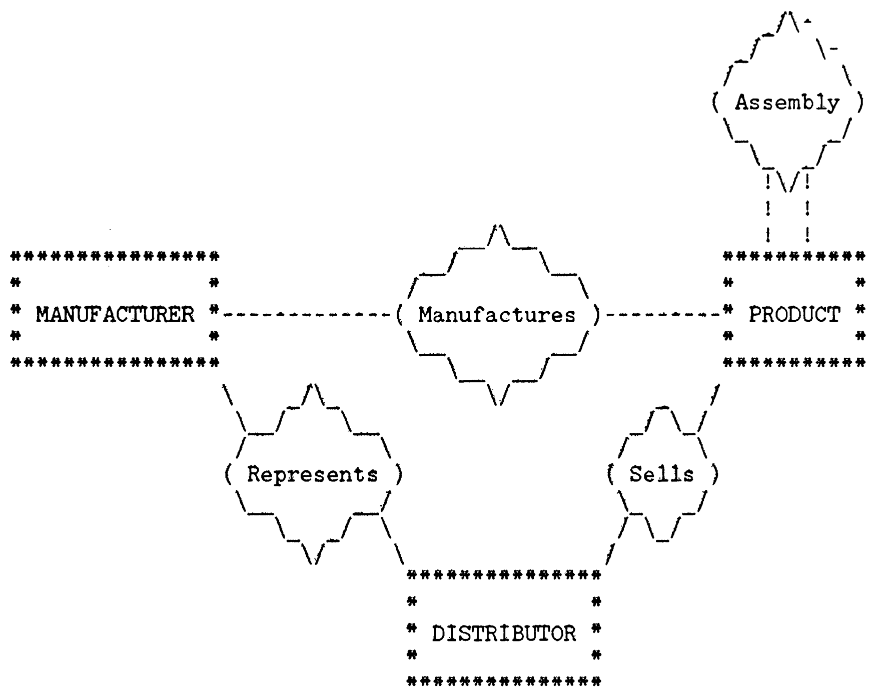
Classify your Entities and your Relationships

Graphics are great for classifying! We like to use RECTANGLES to represent collections of entities, CIRCLES to represent collections of relationships, and lines to indicate WHICH entities are related by WHICH relationships to WHICH other entities.

Since entities and relationships are equivalent, this is a valid choice of geometric figures: After all, a rectangle and a circle are topologically equivalent!

I had difficulties using the standard ASCII characters (not the best pixels in the world) to simulate circles. These blackberry-like things were the best I could do! So, please bear with me...

Regardless of the graphics you use to guide your classification, your entities and your relationships will conveniently fall into categories which are obvious to you. For instance, if you are a manufacturer or a distributor, you probably would choose something along these lines:



Notice, with pleasure, that IMAGE/3000 relates entities in all kinds of arrangements which are absolutely natural and commonplace to you.

Please ignore the fact that these unexpected features may throw some database authorities out of balance! For example, a distributor may represent some manufacturer(s) and still sell products made by OTHER manufacturers! While some theoreticians AGONIZE over this minor issue, IMAGE/3000 simply goes ahead and HELPS YOU IMPLEMENT IT.

As an interesting example of a bill-of-materials, elegantly modeled with the minimum of elements, please see the "Assembly" relationship which relates "products" to "products" so that we may quickly answer either of these questions with equal ease: "Which products do I need to assemble THIS product?" and "Which products can I assemble with THIS product?".

Another example models presidential administrations (or periods):

```
                         ********
                         *      *
                         * YEAR *
                         *      *
                         ********
                           !  !
                           !  !
                           !  !
                          _!__!_
 ***********            __/      \__           *************
 *         *           /            \          *           *
 * COUNTRY *          / Presidential \         * PRESIDENT *
 *         *-------( Administration )-------*           *
 ***********          \              /         *************
                       \__        __/
                          \______/
```

Notice that this design does NOT include restrictions such as "citizenship" and "uniqueness". The same person could be the president of more than one country at the same time. Many persons could be simultaneous presidents of the same country (have you heard recently of "presidents in exile"?). You can find an administration by beginning year or by ending year.

Without any radical changes, this same design could apply to directors of corporations (and would be very useful to trace interlocking boards of directors!)

These apparently complex relationships are a delight to reflect with IMAGE/3000, thanks to its conceptual clarity and modeling power. Ironically, if you look at a average database book, you will see these kinds of relationships used as examples to show what is wrong with average database management systems. IMAGE users of the world: Count your blessings!

Presto

Translate your nice graphics to IMAGE's database definition language. Rectangles ("collections of entities") translate immediately to MASTER datasets. Circles ("collections of relationships") translate immediately to DETAIL datasets. Lines which represent obviously "hot" relationships translate immediately into PATHS.

For examples of syntactically-correct IMAGE/3000 schemas, see Appendix A for the DISTRIBUTOR database and appendix B for the PRESIDENTIAL database.

Refine your indexing for performance

IMAGE's "search fields" just happen to be convenient for the sake of IMAGE's hashing algorithm (which converts a data value to an address) and IMAGE's chaining algorithm (which links logical neighbors even when they are millions of entries away from each other). But you can design ANY mathematical mapping of your choice that will convert any data value into a reference to whatever key you defined for IMAGE.

For instance, I like to use an indexing scheme that I developed with Ross Scroggs back in early 1981 during one of his visits to Guatemala. We used soundex-like algorithms to build powerful structures which allow us very quick answers to questions like "Give me all the Fred's who live in New York City and have a pre-release version of Adager's ItemChng". The fun part of our indexing is that we can get the same answer even if we pose a question which uses the equivalent attributes: "Give me all the Freddies who live in The Big Apple and have product JC810312".

Orchestrate your Transactions

This is the dymanic part of the database! Specify the transactions that will allow you to add, modify, delete, and report these entities and their relationships. Decide whether or not some of these transactions need to be undisturbed by other concurrent transactions. Take advantage of IMAGE's locking to make sure that you achieve a fair compromise between "privacy" and "sharing".

Perform your Transactions

At your convenience, add, delete, find, modify, relate, and report entries. Do it solo or invite all your friends and fellow workers, from the next desk, from the next building, from the next country, or from anywhere in the network. IMAGE/3000 is a multi-tasking multi-computer database management system, after all!

Tune up your Performance

As you specify your masters, your details, and your paths, keep in mind that the important question is: "Can you define, redefine or cancel these entities and their relationships at any time during the life of the database?" For performance reasons, you may want to wire some OBVIOUS relationships "hot" in the database's structure by means of PATHS. But you do not want to be stuck for life, since some hot relationships may cool off and some sleepers may wake up unexpectedly!

Fine tune things in such a way that you reach a reasonable compromise between the RESPONSE TIME for any of these functions and the global THROUGHPUT for the whole transaction load.

Bravo! You are now a Database Maestro, thanks to IMAGE/3000.

Appendix A

IMAGE/3000 schema for the DISTRIBUTOR database mentioned in the "Practical Methodology" section.

```
Begin database DISTR;


<<

NOTES:  YOUR imagination and convenience should decide how many (and
        which) attributes to include at the "...".

        The paths are NOT necessary at all, but we include them as
        examples of performance boosters for relationships which
        seem to be "hot and heavy".  You can always take ALL paths
        away, or add OTHER paths at will.  IMAGE/3000 does not care!

        Capacities can go from 1 to several million.  The help of
        intelligently-defined paths becomes more obvious when you
        deal with millions of entries.  Toy-like academic examples,
        of course, do not require any overhead in terms of structure.

>>


Passwords:

  10  SeeAll;
  <<...>>


Items:

  Manufacturer#,  x6 ;
  Distributor#,   x4 ;
  Product#,       x10;
  assembly,       x10;
  component,      x10;
  name,           x40;
  address1,       x40;
  address2,       x40;
  city,           x30;
  department,     x30;  <<Some countries may use "State" or "Province">>
  country,        x30;
  amount,         r4 ;
  production,     j2 ;
  supervision,    j2 ;
  responsibility, j2 ;
  <<...>>


Sets:

Name:  MANUFACTURER, manual;

Entry:
  Manufacturer# (2),  <<paths to "Manufactures" and "Represents">>
  name,
  address1,
  address2,
  city,
  department,
  country;
  <<...>>

Capacity: 2000;


Name:  PRODUCT, manual;

Entry:
  Product# (4),  <<paths to "Manufactures", to "Sells", and two paths
                 to "Assembly">>
  name;
  <<...>>

Capacity: 80000;


Name:  DISTRIBUTOR, manual;

Entry:
  Distributor# (2),  <<paths to "Represents" and "Sells">>
  name,
  address1,
  address2,
  city,
  department,
  country;
  <<...>>

Capacity: 20000;


Name:  Assembly, detail;  <<Relates products which are, in turn,
                            parts of other products>>

Entry:
  assembly  (PRODUCT),  <<This search field allows us to find all
```

```
                          the products which we need to assemble a
                          given product>>

  component (PRODUCT),  <<This search field allows us to find all
                          the products which we can assemble with
                          a given product>>

  amount,               <<of the component product in the assembly>>
  Production,           <<Person in charge, for instance>>
  Supervision,          <<Person in charge, for instance>>
  Responsibility;       <<Person in charge, for instance>>
  <<...>>

Capacity: 150000;


Name:  Manufactures, detail;  <<Relates manufacturers to products>>

Entry:
  Manufacturer# (MANUFACTURER),
  Product#      (PRODUCT);
  <<...>>

Capacity: 2000000;


Name:  Represents, detail;  <<Relates manufacturers to distributors>>

Entry:
  Manufacturer# (MANUFACTURER),
  Distributor#  (DISTRIBUTOR);
  <<...>>

Capacity: 5000;


Name:  Sells, detail;  <<Relates distributors to products>>

Entry:
  Distributor#  (DISTRIBUTOR),
  Product#      (PRODUCT);
  <<...>>

Capacity: 5000;


End.
```

Appendix B

IMAGE/3000 schema for the PRESIDENTIAL database mentioned in the "Practical Methodology" section.

```
Begin database CHIEF;


<<

NOTES:  YOUR imagination and convenience should decide how many (and
        which) attributes to include at the "...".

        The paths are NOT necessary at all, but we include them as
        examples of performance boosters for relationships which
        seem to be "hot and heavy". You can always take ALL paths
        away, or add OTHER paths at will. IMAGE/3000 does not care!

        Capacities can go from 1 to several million. The help of
        intelligently-defined paths becomes more obvious when you
        deal with millions of entries. Toy-like academic examples,
        of course, do not require any overhead in terms of structure.

>>



Passwords:

  35  GuessWho;
  <<...>>


Items:

  Country#,        i  ;
  President#,      i2 ;
  Name,           x50;
  Year,            i2 ;
  Year-In,         i2 ;
  Year-Out,        i2 ;
  party,          x20;
  <<...>>


Sets:
```

```
Name:  COUNTRY, manual;

Entry:
  Country# (1),  <<path to "Administration">>
  name;
  <<...>>

Capacity: 300;


Name:  YEAR, manual;

Entry:
  Year (2);  <<two paths to "Administration">>
  <<...>>

Capacity: 4000;  <<You might want to go back to the Etruscans!>>


Name:  PRESIDENT, manual;

Entry:
  President# (1),  <<path to "Administration">>
  name;
  <<...>>

Capacity: 20000;


Name:  Administration, detail;  <<Relates countries to presidents to
                                  In and Out years>>

Entry:
  Year-In  (YEAR),  <<This search field allows us to find all
                      the administrations, anywhere in the world,
                      which began in a given year>>

  Year-Out (YEAR),  <<This search field allows us to find all the
                      administrations, anywhere in the world, which
                      ended in a given year>>

  Country# (COUNTRY),  <<This search field allows us to find all the
                         administrations for a given country>>

  President# (PRESIDENT),
  Party;               <<Who knows?  Presidents might have different
                         parties from one administration to the
                         other.  Party is NOT really an attribute of
                         PRESIDENTS, as it might appear.  Party IS an
                         attribute of ADMINISTRATIONS!>>
  <<...>>

Capacity: 150000;
```

End.

=========

Gratitude

=========

Rene Woc (partner in business and science) and Leslie Keffer de Rego (partner in life) helped me refine the ideas and the presentation.

=========

Biography

=========

F. Alfredo Rego is Adager's Research & Development Manager. He has worked with Hewlett-Packard instruments since 1966, when he was a Physics research assistant in the Center for Nuclear Studies at The University of Texas. In the 1970's he worked as a university professor in Guatemala, teaching courses in Theoretical Mathematics, Physics and Computer Science. He has worked exclusively with IMAGE databases and Adager (The Adapter/Manager for IMAGE/3000 Databases) since 1978.

The HP3000 International Users Group honored him with the 1980 Hall of Fame Award, which reads: "Outstanding Contributor, for exemplary service to the Group and its membership".

3076. WRITING EFFICIENT PROGRAMS IN FORTRAN/77 ON THE HP 3000

Carolyn Bircher
Hewlett-Packard, Cupertino, California

INTRODUCTION

The most important step to be taken when approaching the problem of program efficiency is to select the most appropriate algorithm for the problem to be solved. The programming optimizations discussed in this paper may be insignificant compared to the gains which can be achieved by the improvement in the overall solution to the application. The choice of algorithm should be determined by many factors, such as the resources available and the amount and characteristics of the data to be processed.

Beyond the algorithm, though, significant steps may be taken to further optimize your program. By manipulating your source, you can enhance your program to perform its current algorithm most efficiently. By having an understanding of the internals of the FORTRAN compiler and of the machine code that it generates, you will be able to minimize the amount of work required to execute your program.

This paper will present many of the places where modifications can be made which will decrease either execution time or data space usage in FORTRAN/77 programs. Although it was written specifically about the FORTRAN/77 compiler on the HP 3000, many of the optimizations would be relevant on any system.

Five major areas of the FORTRAN language will be addressed. They are data declaration, expressions, control structures, program unit interfacing, and I/O.

DATA DECLARATION

- Select Most Efficient Data Types

When you have a choice, use the most efficient data type. Integers are the most efficient, followed by real, and double precision is the most costly. Use real and double precision only where you need the ability to handle fractions or very large values.

Timings done on an HP 3000/series 44 showed the following differences in the time required to do arithmetic operations. The times are measured in milli-seconds and some are approximations.

Operation	INTEGER*2	INTEGER*4	REAL*4	REAL*8
Add/Subtract	.840	.840	5.780	20.800
Multiply	3.203	6.773	9.555	35.500
Divide	3.780	15.330	12.075	44.400

Table 1

Your program will generally run most efficiently if you set INTEGER and LOGICAL sizes to equal the word size of the machine on which you are running. Since the 3000 is a 16-bit computer, you should use the $SHORT compiler option to change the default sizes to INTEGER*2 and LOGICAL*2. This will also save space on your data stack since integers and logicals will only take up half as much space.

If you are using a 32-bit machine, such as the HP 9000, let the defaults remain at INTEGER*4 and LOGICAL*4.

- Avoid Indirect Addressing

Avoid the use of arrays, equivalenced data, common variables, and variables with a length greater than 64 bits. These elements are all addressed indirectly. That is their address must be found first and then the element at that address is found. Indirect loads and stores take more than 50% longer than direct.

Use the 3000 $MORECOM directive only in programs where you cannot avoid the use of a very large number of COMMON variables. This option introduces an additional level of indirection when addressing common variables.

You can sometimes increase the number of variables allowed in a COMMON block on the HP 3000, without using the less efficient MORECOM option, by changing the order of the variables. Each COMMON block element normally has one pointer established by the segmenter. The maximum number of pointers allowed is 254 when MORECOM is not used. Therefore, there would be three pointers for a COMMON block containing the variables C(3),A,B.

However, if the order of the variables is changed to A,B,C(3) (the array following the simple variables), only two pointers are required. This is because array pointers are set to point to the zero'th element of the array. Since arrays start with element one in FORTRAN, the zero'th element would be at the same location as B when the variables are listed A,B,C(3). Therefore, a single pointer will be shared for variable B and array C.

- Minimize the Size of the Data Stack

Avoid the use of static variables. COMMON variables, variables initialized in DATA statements, and SAVEd variables all will remain on your data stack throughout the run of your program. Non-initialized local variables will be there only when the subroutine in which they are declared is active.

Do not place variables that are accessed by only one routine into COMMON blocks. If a variable is used by only a couple of routines, it may be advantageous to pass it between the routines as a parameter instead of putting it into a COMMON block.

When including character variables with an odd number of characters in a COMMON or EQUIVALENCE, group them together.

```
$SHORT
      CHARACTER*3 C3, D3
      CHARACTER*5 C5, D5

      COMMON /C1/ C3, I1, C5, I2
      COMMON /C2/ D3, D5, J1, J2

```

When we prep this program, the PMAP information shows that the COMMON named C1 takes one more word of storage than C2 even though they contain the same data types. This is because a byte is wasted after each of the character variables so the integers will be alligned on a word boundary.

```
COMMON ARRAY ALLOCATION

   NAME                ADR   LEN
   C1                   10     7
   C2                   17     6
```

- Initialize Data Efficiently

Use DATA statements to initialize global variable values. On the 3000, all variables included in DATA statements are initialized by a single move when the program is loaded, so code does not have to be executed to initialize each one separately. However, as mentioned above, variables which are initialized in DATA statements are kept on your data stack throughout the entire run of your program, so you probably do not want to use the DATA statement to initialize local variables. Also, the data statement only initializes at the beginning of program execution, not each time that the subroutine that the variable resides in is executed.

EXPRESSIONS

- Do Not Mix Data Types

When expressions contain more than one data type, one or more of the variables must be converted. The conversion is always to the less efficient data type. For example, the assignment statement J=1.0+J, where J is an integer, requires conversion of J to a real and then conversion of the result back to integer. A more efficient assignment statement would be J=1+J.

If it is necessary to use different data types in an expression, group the most efficient type at the left. Operations at the same precedence level are evaluated from left to right so the leftmost operators will be evaluated first and the result will then be converted to the data type required for the remainder of the expression. In the expression R = R + I1 + I2 both I1 and I2 will be converted to reals and floating point adds will be done. If the expression is rearranged to R = I1 + I2 + R, the two integers will be added together and then the sum will be converted to real and added to R.

This could also be achieved by using parenthesis to group the variables. R = R + (I1 + I2) would be just as efficient as R = I1 + I2 + R. This makes it a little more obvious that the variable grouping is intentional. It is also a good idea to use the conversion intrinsics to make it really obvious that a conversion is taking place. The above expression would be written as R = R + FLOAT(I1 + I2).

- Reduce Strength of Operations

As you saw in Table 1, some arithmetic operations are faster than others, even within a single data type. It is almost universally true among computers and data types that the amount of time required to execute arithmetic operations occurs in the following order.

```
1. Addition (+) and Subtraction (-)    <- fastest
2. Multiplication (*)
3. Division (/)
4. Exponentiation (**)                 <- slowest
```

When possible, you should reduce the strength of the operations in your expressions. Exponentiation can often be replaced by a series of multiplies (j**2 becomes J*J). When multiplying by an integer constant, you could replace the multiply with a series of adds (J*3 would become J+J+J). When dividing by a constant, you sometimes can transform the expression into a multiply by a fraction (Y/10 becomes Y*0.1).

By applying some algebraic rules, you can also find opportunities for reducing operation strength in expressions. For instance, the series of divides A/B/C/D could be replaced by A/(B*C*D).

However, be aware of the fact that when these transformations are done in expressions involving real data, the results may be slightly different, due to floating point rounding differences.

- Enable Constant Folding

If you include more than one constant in an expression, try to group them at the beginning. The compiler evaluates expressions from left to right on each precedence level. As long as it has found only constants, it will fold them. As soon as a variable has been found, though, no more folding will be done. If a program contains:

```
PARAMETER (MIN_PER_HOUR=60, SEC_PER_MIN=60)
INTEGER SECONDS, HOURS, MIN_PER_HOUR, SEC_PER_MIN
...
SECONDS = MIN_PER_HOUR * SEC_PER_MIN * HOURS
```

the compiler will generate code as if the expression had been written 3600 * HOURS. But if the variable, HOURS, had been placed between or before the two constants, a separate multiply will be done for each constant. You could also effect the order of evaluation by using parenthesis to group constants.

- Avoid Unnecessary Blank Filling

When assigning character data, blank filling will occur if the target variable is longer than the data being assigned to it. Extra code will be generated to do the blank filling when assigning a variable. When assigning a character constant, the blanks required to fill the remainder of the target are appended to the string when it is put into your code segment.

If blank filling is not required for your application, you can avoid either of these inefficiencies by assigning to a substring which is equal to the length of the string being assigned. The statement CHARSTRING='ABC' could be rewritten as CHARSTRING(1:3)='ABC'.

CONTROL STRUCTURES

- Eliminate Invariant Code in Loops

Do not put invariant code within the scope of a loop. If the result of an operation will be the same every time through the loop, put it outside the loop so it will only be executed once.

- Minimize Loop Overhead

There is a significant amount of overhead involved in executing a DO loop. Before starting the loop, several variables must be initialized. At the end of each loop interation, the loop control variable must be incremented and compared to the end value.

You can eliminate some loop overhead by combining adjacent loops with similar loop controls. The loops

```
         DO 100 I = 1,20
     100   A(I) = B(I) + C(I)

         DO 200 J = 2,40,2
     200   X(J) = FLOAT(J)
```

could be combined into the loop:

```
         DO 100 I = 1,20
           A(I) = B(I) + C(I)
           X(I+I) = FLOAT(I+I)
     100 CONTINUE
```

If a loop is going to be executed a small, fixed number of times you can eliminate loop overhead by rewriting the loop as several separate statements. This is worth doing if there are not many statements within the body of the loop and it is only executed a few times. This loop

```
         DO 100 I = 1,4
     100   A(I) = FLOAT(I)
```

could be replaced by

```
         A(1) = 1.0
         A(2) = 2.0
         A(3) = 3.0
         A(4) = 4.0
```

- Use Efficient Loop Indexes

On the 3000, the MTBA (modify variable, test and branch) instruction makes it possible to implement DO loops with an INTEGER*2 index variable considerably more efficiently than loops using other data types. If you have a choice, make sure that the loop index is an INTEGER*2 variable. The overhead for MTBA loops is only 1 instruction at the end of each loop iteration compared to 10 instructions for an INTEGER*4 loop. Note, however, that this instruction will not be generated in subroutines in which assigned GOTO's are included.

Do not use REAL or DOUBLE PRECISION loop indexes. The index is incremented by the step value and compared to the end value each time through the loop. These adds and compares are done in the same data type as the loop index.

- Order Logical Conditions Efficiently

When writing complex conditions in IF or WHILE statements you can minimize the time required to execute them by rearranging the order in which they occur. In block IF statements, order the conditions so that the most likely condition is tested first. For example, if the value of ARG is 3 in most cases, write a compound IF statement as:

```
IF (ARG .EQ. 3) THEN
   ...
ELSE IF (ARG .EQ. 1) THEN
   ...
ELSE IF (ARG .EQ. 2) THEN
   ...
ELSE
   ...
ENDIF
```

When using the logical operators .AND and .OR. in a logical expression, the code generated only checks enough conditions to determine the result of the entire expression. The conditions are checked from left to right, with AND having precedence over OR. If several logical expressions are connected with .OR., checking discontinues as soon as an expression is found which evaluates to .TRUE. When .AND. is used, checking discontinues as soon as a .FALSE. condition is found. When writing logical expressions, you should order the conditions so the least number of checks is done.

If variable I is more likely to equal zero than variable J, write the IF statement as

```
      IF ((I .EQ. 0) .OR. (J .EQ  0)) ...
   or
      IF ((J .EQ. 0) .AND. (I .EQ. 0)) ...
```

When assigning to a logical variable, use a simple assignment statement instead of an IF/THEN/ELSE block. Instead of writing

```
      IF (A .EQ. 0) THEN
         L = .TRUE.
      ELSE
         L = .FALSE.
      ENDIF
   use
      L = (A .EQ. 0)
```

- Do Not Use Range Checking

Turn $RANGE checking on only when absolutely necessary. This directive causes extra code to be included in your program to check the bounds whenever a substring or array element is referenced. Code is also generated for checking assigned GOTO's, DO loop ranges, and parameters passed to some intrinsics. These checks could be invaluable while you are debugging your program, but you probably do not want to pay the performance penalty for them indefinitely.

PROGRAM UNIT INTERFACING

- Avoid Unnecessary Procedure Calls

There is a lot to be said for modular programming. It makes programs easier to read and to write. But each time a branch is made to a subroutine, you pay the overhead of loading the argument addresses and the stack marker onto the data stack and of doing the branch.

One program modification which can be made to eliminate procedure call overhead without sacrificing modularity is to replace short function subroutines with STATEMENT FUNCTIONs. Instead of doing a branch, the code to execute a statement function is expanded each place where the function is referenced. Of course, doing this will make your program file larger but it will run faster.

Something else to keep in mind is that the compiler generates procedure calls to execute some arithmetic operations. If you can, eliminate those procedure calls by reducing the strength of those operations. Exponentiation is one such operation so the statement X=Y**2 should be rewritten as X=Y*Y to avoid the external call.

- Segment Efficiently

Segment your program with efficiency in mind. If a large amount of interaction will take place between two program units, make sure that they are placed in the same segment. Try to minimize the total number of segments used, without making any of them too large. The PMAP listing generated by the segmenter will show you where calls are being done across segment boundaries.

- Use System Programming Language

In some instances it may be worth your while to write part of your program in the system language for your machine. One example of this would be when moving an entire array to another array with like dimensions. In SPL you can do the move as a single instruction instead of moving each array element separately. If the array is large, this could save a significant amount of execution time.

I/O

- Format Only When Necessary

Use formatted I/O only when reading or writing data which must be seen by the human eye.READ and WRITE statements which reference a format string generate calls to the data formatter which translates the data between ASCII and the internal format. On top of the overhead for this procedure call, the conversion is very costly and can cause a loss of precision in real data.

When you must use formatted I/O, do not put the format string into a variable. Variable format specifiers are not parsed when the program is compiled. Instead, the parsing routine is called each time the format is used. These statements should be used

```
      WRITE (6,20) VAR
   20 FORMAT (F10.2)
```

instead of

```
      CHARACTER*7 FMT
      FMT = '(F10.2)'
      WRITE (6,FMT) VAR
```

- Use System I/O

Use MPE intrinsic I/O instead of FORTRAN READ and WRITE statements. The FORTRAN statements generate several procedure calls for each statement. For instance, the statement

```
      READ (IUNIT) I,J,K
```

would generate calls to five I/O library routines which in turn call other routines.

- Operate on Entire Arrays

When reading or writing an array, specify just the array name instead of using an implied DO loop. This allows the array to be operated on as a whole, instead of performing individual operations for each element. For example, specify

```
      WRITE(6) MYARRAY
```

instead of

```
      WRITE(6) (MYARRAY(I,J), I=1,10), J=1,10)
```

COSTS OF OPTIMIZATION

There is no such thing as a free lunch. Before you rush out to begin optimizing your FORTRAN/77 programs, I feel obligated to caution you about some of the costs of optimizing.

Many of the optimizations mentioned above require that you reorder or rewrite expressions. Although those kinds of changes may be easy to make, they may also make the intended purpose of the expression less obvious to someone reading your source code. Be sure to carefully document any optimization changes that you make.

Also, when optimizing, you may be trading one kind of efficiency for another. A change that will decrease execution time may also increase the amount of space required for your data or your code. Several of these trade-offs were mentioned above.

Finally, it is sometimes difficult to justify the programming time required to optimize. Performance evaluation usually gets left to the last of a software project and when time constraints become too tight, it may get skipped. Once you start making the kinds of changes discussed in this paper, many of them will become second nature and you will be able to build them in as you do new development. But you will still probably want to allow time in your schedule to evaluate and improve program performance.

If you find that the time available for optimization is limited, you should concentrate your effort on the parts of your program where you will get the best return on your investment. Studies have shown that typically 90% of a program's execution time is spent in 10% of its code. Performance measurement tools, such as SAMPLER/3000 (product number 32180A) are helpful for determining where your program's execution time is spent.

BIBLIOGRAPHY

Hewlett Packard, FORTRAN/77 Programmer's Guide, (1985), Part No. 5957-4686.

Metcalf, Michael, FORTRAN Optimization, Academic Press, London, 1982.

BIOGRAPHY

Carolyn Bircher is a Software Development Engineer in the Computer Language Lab at the Hewlett-Packard facility in Cupertino, California. She joined HP in 1979 after receiving a B.S. degree in Computer Science from California Polytechnic State University at San Luis Obispo. She is currently on the FORTRAN project team which has responsibility for the FORTRAN/77 compilers on the HP 3000, the HP 9000/series 500, and future HP systems.

3077. Operational Considerations for Police Networks

Jerry Kopecky
Illinois Criminal Justice Information Authority
120 South Riverside Plaza
Chicago, Illinois 60606

There are many different aspects involved in designing, implementing, and then successfully running police systems. The Illinois Criminal Justice Information Authority has been operating police systems since 1981. Operational considerations involve 24-hour uptime (system backup, auditing, security issues), hardware selection, software design, online response time, management report requirements, remote site support, and operator training. These operational considerations are perhaps common to other businesses but reflect additional police requirements. 1981. Their underlying premise is one basic tenet: information is a common resource shared not only within individual police departments but also among other departments, and other components of the criminal justice system.

Introduction

The Illinois Criminal Justice Information Authority has nearly a decade of experience in developing computerized management information systems for law enforcement agencies, prosecutors, and correctional institutions. From a rather inconspicuous beginning as a small research and development unit, the Authority has evolved into a state agency that is legislatively mandated to develop and operate computer systems for the criminal justice community in Illinois. Not coincidental with this expansion, the Authority has become very adept at applying new and emerging technologies to the information problems confronting criminal justice practitioners.

The underlying strategy behind the creation of the Authority was to provide a forum of criminal justice professionals who could identify information problems and requirements: and provide staff support to address these problems in a timely and efficient manner. The Authority is made up of a 15 member board appointed by the Governor, and consists of key criminal justice administrators, representing state, county and local agencies. Staff to the Authority are organized into five different divisions (see Figure 1).

- o Office of the Executive Director - the Executive Director is appointed by the Governor and is responsible for the administration of the Authority.

- o Personnel and Budget - which is responsible for personnel transactions, payroll, budgets, and contracts.

- o Policy and Research Division - which is responsible for fulfilling our mandate with respect to research, policy development, program coordination, and information correlation.

- o Office of Federal Assistance Program - which is responsible for the coordination of Federal criminal justice programs.

- o Information Technology Division - This division consists of two functional centers: Computer Operations Center, responsible for the support and maintenance of our computer lab facility, and the Systems Development Center, which is responsible for the development, implementation and maintenance of software developed by the Authority for use other agencies.

Police Information Management System - PIMS Configuration

One of the largest and most resource intensive applications developed and operated by the Authority is the Police Information Management System (PIMS). PIMS is a computerized management information system designed to increase a police department's tactical and strategic effectiveness and to perform such paper-intensive tasks as maintaining police records and producing management information for administrators. PIMS is one component of the Authority's Criminal Justice Information System (or CJIS), a set of computerized information systems for law enforcement, prosecution and correctional agencies. Although PIMS can operate by itself, jurisdictions can realize additional benefits when PIMS information is used in conjunction with automated prosecution and correctional data.

The Authority provides central site computing facilities for a range of different criminal justice agencies. Current hardware consists of two Series 68s, one 48, and one 42 (see Figure 2 for the ICJIA network). The PIMS network provides service to 22 different communities (population of 3/4 million people located in the suburban Chicago area and uses the two 68s.

The most important requirement of a police system is to be available 7 days a week/24 hours a day. The system cannot be down for more than 20 minutes at any point during the day. Hewlett-Packard maintenance support policies require that the 3000s be available for preventative maintenance sessions several times a year. For the equipment configuration of the Authority, this requires about 3-5 hours for each PM session. For those times when hardware failures occur, even the shortest time for a repair takes 1 1/2 hours (from the time the call is placed to when the repair has taken place). The longest time experienced for a repair has been 3 days.

Therefore, our "uptime" requirement necessitates the availability of a second machine with a "mirror" configuration to serve as a "hot" backup to the primary production machine. This second machine is appropriately named Backup. It is also a Series 68 with the same amount of memory, disc drives, tape drives, ATPs, etc.. It also has the same I/O configuration as the Production system (see Figure 3).

When the Production system experiences a hardware failure or a preventive maintenance session is scheduled, it takes Operations staff approximately 15-20 minutes to completely switch the online PIMS' applications to the Backup system.

Hewlett-Packard does not offer redundant systems or shared peripherals between different CPUs so the Authority uses MPE's private volume facility to switch the data on H-P 7935 disc packs from one system to the other. This obviously requires the account structures to be identical on each system. This includes everything from volume sets to group and user names. All application data and programs are configured to reside on the 7935 disc drives. The longest part of the entire switch process takes place when the drive is switched on after the pack has been inserted. Each 7935 disc drive takes approximately 7 minutes to go online. All system software including various software utilities are configured to reside in the system domain on one H-P 7925 disc drive.

To make the switch, several procedures are always followed. The first procedure is to warn the currently logged on users that the system is coming down (obviously, this step is not needed if the system has already crashed!). When the users and the application programs have logged off, the private volume sets are logically dismounted with the various console commands and the private volume domain disc drives are brought down. The disc packs are then removed from one system and moved to the other. In addition, the packs are wiped down with a cleaning solution before they are inserted into the disc drives. This additional procedure has significantly reduced the number of bad sectors and tracks on the various packs.

One common "glitch" noticed when the drives are brought back online is the message "DISC FREE SPACE MANAGEMENT ERROR". The H-P recommended procedure at this point is to do a coolstart and recover lost disc space. However, this takes another 10 minutes. Instead, the procedure is to quickly take the drive offline and then put the drive back online. This usually does the trick and is much faster than a coolstart.

The Authority uses very simple mechanical switches to switch the other peripheral equipment. A H-P 1000 computer system is currently used as a Front End processor to the Series 68. The 1000 has all the terminals and printers hardwired to it so to switch systems, only one switch has to operated. The new version of PIMS has eliminated the Front End so all the terminals and printers are now hardwired directly to the Series 68. A series of mechanical switches now switch from one system to the other. This series of switches was custom designed so that only nine switches have to be operated to completely switch up to 250 devices. The switches are contained in one standard 19 inch bay and require no electrical power.

After the packs and peripherals have been switched, various application programs are fired up. These programs include bringing up various communication lines (DS and IMF), data bases, and informing users what the current status is.

The redundant systems allow the Authority to offer better than 99% system availability to its users. The 99% figure still means there are periods of down-time. For instance, in a one month period, 99% means there is anywhere from 7-10 hours of total down-time. Effort to reduce this is constantly being worked on.

System Performance Optimization

In order to maximize the efficiency of the system, modules of the application have been segregrated. When the Backup system is not in the "switched" mode, efficiency dictates its use. The PIMS network offers the feature of providing additional capabilities to the police system users that are not as critical and sensitive on the Backup system. The additional capabilities do not have to be constantly available to the user.

The Authority has integrated online transactions with batch reporting functions yet segregated the two on two different machines. Batch and online tasks have very different requirements that make it difficult to "tune" and optimize one system to efficiently and effectively process both functions. Therefore, the functions have been split to the two machines.

PIMS offers two batch reporting functions: the Management Reporting module and the Search module, both run on the Backup system. The data for these modules are a subset of the data from Production data bases and reside in KSAM file format. The Backup KSAM files get updated daily from the IMAGE logfiles taken from the Production system.

The Management Reporting module produces management reports on the Backup 68. These reports are written using COGNOS's QUIZ report package. Currently, there are approximately 200 reports available to the user on demand. All reports are requested "as needed, online" from the Production system by the individual police departments. The report is then submitted to the Production machine's job queue. When the job logs on, the JCL statements are transferred to the Backup system via a dsline. The job then logs on Backup and executes. When finished, the job JCL transfer the print file back to the Production system where it is produced on a printer located at the department. Since the reporting requirements of one department may vary from those of other agencies, each department controls its own report initiation and specifications.

The search module operates in a similiar manner and is used for creating lists of possible crime suspects and for analyzing particular offense patterns. It lets the user search for and access records even when the user does not have precise information, such as a name or incident number. For example, other modules collect physical descriptions of arrestees, the search module allows the user to retrieve the records of all individuals matching a certain physical description who have been arrested on PIMS for a particular offense. The Search Module also allows users to search for information about similar incidents, property types and crime locations. Figure 4 gives an example of a typical search request.

The Authority has taken a machine that always needs to be available and moved less critical applications to it. The integration of online access and the submission of batch jobs to the second machine do not conflict and compete with each other.

Problem Isolation

At the Authority's central computing site, there are presently 22 police departments connected online via leased phone lines. At each department, there are an average of 7 devices (4 terminals and 3 printers). Figure 5 illustrates a typical department's device layout. All of these sites are at least 15 miles from the central site so all problems are handled over the phone. Since problems are inevitable (and one might add, "normal"), the Authority has come up with a number of techniques for problem isolation and problem resolution.

First Line "Defense"

All of the departments are responsible for their own equipment. This includes terminals, printers, muxes, and telecommunications equipment. They are also responsible for obtaining their own service contracts for this equipment. The Authority provides what is called "first line defense". When a question occurs, the user always calls the central site - henceforth called Operations - first. Operations then makes the determination as to what needs to be done. Problems can be of any nature, ranging from remote hardware, software, tele-comm equipment problems. The other category of user calls are for information. A user may not know how a particular transaction works or what the policy is for items in a transaction.

The rationale for the "first line" is quite simple. Operations usually knows more about what is going on with the entire system at that particular point in time. If there are problems elsewhere in the system, user generally does not have access to that information. Most users are not technically oriented so they do not know how terminals, printers, or modems (since digital leased lines are used, modems are not used but the word remains in the vocabulary) work. Once the problem has been identified, it is either resolved directly by Operations or Operations makes the necessary service calls to the vendor. Vendors appreciate this arrangement because it allows them to better identify the problem and serves to "weed out" unnecessary site visits. The configuration of each site is known and knowledge of the entire system aids with the resolution.

However, the users are a very important component in problem resolution. They serve as very effective "distant warning outposts" to provide advance notification of possible future problems. For instance, a user complaint of slow response time may indicate a master data set is getting full or a frequently used transaction should be examined and perhaps rewritten to work more efficiently.

Operations keeps track of the type of equipment maintenance that each department has contracted with vendors. Staff have to know this information so that unnecessary service (and costly) calls are not made. Because of the 24 hour usage of the equipment by different personnel shifts, even the users do not know what type of service they have contracted for on each piece of their equipment.

Operations keeps several different types of logs. Every user call is tracked by recording several items about each call. Figure 6 illustrates the screen layout of the log. This log is then periodically reviewed by several different layers of people involved with the project, starting with Operations Management and then rolling forward thru project staff and individual department staff. There are usually a large number of software questions that come in. The log gives staff the capability to find out who may need additional training or where bugs or enhancements need to be worked on.

An equipment maintenance log is kept and is periodically reviewed with vendor support staff. This log is invaluable in that it shows where the "lemons" are so that they can quickly be removed from the network.

To help with equipment breakdowns, the Authority has established several geographically diverse "hot" sites that contain equipment to be used as spares. The spares allow for several department to reduce their equipment maintenance costs by sharing the purchase price of backup equipment. The Authority has contracted with the vendor to provide this service at a reduced rate. When problems do occur, Operations directs the police department to drive to the hot site, pick up the spare equipment, and is then led thru the procedures necessary to connect the equipment. The vendor is informed about this during normal business hours and his only task is to make arrangements to pick up the bad unit and replenish the hot site's inventory.

Another "automatic" log that is used is a program that monitors data set capacities and then stores these figures. Simple extrapolation can project when the set will reach a dangerous level or maximum capacity thus allowing the scheduling of time to make necessary adjustments.

The main purpose of these logs are not just to keep a history of problems but to help to possibly eliminate similiar problems in the future. Problems start to quickly show patterns or trends that do not need an intricate analysis to resolve or to predict.

One unique aspect of the police application is the very precise audit trail of user calls. Most of the police departments make tape recordings of all of their phone calls. As a result, they have a very precise and accurate description of how things were handled. Operations staff are very cognizant of any wrong or misleading information they may give to any user. This type of information is very helpful in reconstructing sequences of events or situations.

Another important consideration with problem resolution is the very mundane task of keeping the user informed about what is going on. It is very basic, yet can become complex when the number of users involved becomes high. To aid with this simple task, several standard "canned" JCL files are used that have the necessary session names included so the file can quickly be opened (via QEdit) and then streamed. On an individual department basis, periodic status reports are always made to the user, informing them of such things as vendor arrival time or some fix to their problem.

"The Talking Box"

The Authority maintains and supports several different applications besides the PIMS network. Included in this are the EDP functions of the Authority, court, prosecutors, and jails. As a result, staff do not constantly monitor the various consoles of the police systems. They generally are not even in the computer room.

To help with this, a Digital Pathways SLCII "talking box" was purchased. This device monitors console lines and when certain phrases appear on the console, loud phrases are spoken by the box. A time clock is then started. If the box is not reset, at the end of the time limit, a phone list of numbers is called with the SLCII auto dial capability. The first number called is the computer room's phone number and then people's home phones. The message spoken warns that there is a problem and it has not been resolved. If nobody answers their phones, the SLCII goes back to the top of the list and starts calling the list again. Figure 7 illustrates how the SLCII is connected.

Backup Strategy

A tape backup policy has evolved over a period of time. Regular DBSTORE tapes are made of the data bases twice a week. Currently, it takes approximately 1 1/2 hours for each DBSTORE job. In addition, IMAGE logging is used for recovery purposes. Every day at noon, the PIMS data bases are unavailable for about 7 minutes (a five minute warning of the pending store is given). This time period is necessary to stop the logging process and switch to a new empty log file. The log file is then stored on tape. This IMAGE log file is then used to keep the subset portion of the data current over on the Backup system. The new Turbo-IMAGE log features should eliminate this down time.

There are portions of PIMS that never go down so they "never" get backed up. There is a module of PIMS that communicates with the State of Illinois' Department of State Police's LEADS system. This system serves as the state's central repository for all wants, warrants, car, and driver information. LEADS also serves as a connecting switch to the National Crime Information Computer system operated by the FBI in Virginia. The H-P Series 68s serve just a switch to the LEADS system. The software necessary for this communications switch uses a very small configuration data base and message files to route both incoming and outgoing data. This module of PIMS is very heavily used by the police radio dispatchers when they communicate to patrol cars out in the field. As a result, the LEADS module is the most sensitive to any down time and the critical module that must always remain up.

The tapes used for the data base backup enter a cycle where they are stored in several different places. For two weeks they are kept in a tape room easily accessible to the computer room. Then they are moved to room sized steel vault on the premises for another two weeks. They are then moved off-site to safe deposit vaults located in a commercial bank. The tapes are kept here for another 5 months before they are then brought back to the site and re-cycled. The tapes made for the week of January 1 and July 1 of each year are not re-cycled and are kept for archival purposes.

Other tapes that receive similiar treatment are the tapes made for software releases. Each time software is released, a copy is kept on site and a second copy moved to the off site location. Two versions of software release tapes are kept - the current version and the previous version.

Conclusion

Participating departments share the cost of operating and maintaining PIMS, thus keeping their expenses down. Authority staff develop and maintain all PIMS programs and provide necessary Operational support. The Authority assesses a monthly user's fee to each department: the exact amount depends on the department's size and police activity. This user's fee covers only a portion of the costs associated with operating and maintaining the system. It helps pay for such items as 24-hour-a-day computer service, central site hardware maintenance, and utilities. Software development costs are paid for with funds appropriated by the State of Illinois, and additional support for the system comes from the sales of PIMS software systems to agencies outside the Illinois.

The most important and unique feature of PIMS is that the user department not only has access to its own records but is able to share valuable law enforcement data with other departments. The system provides local police departments with centralized data processing and EDP resources for hardware, software, and technical consulting. However, as PIMS grows and expands, distributed data processing will be necessary to provide adequate network support. Distributed data processing is feasible due to the modular design of PIMS.

The demand for PIMS by Illinois police departments has necessitated the system to be distributed. There are only so many devices that can be connected to a 3000 before response time degradation becomes a problem. The modular nature of PIMS lends itself to be very easily distributed over several 3000 systems located at individual police departments. Declining hardware costs have made owning a small computer more affordable. However, the cost of providing operational support and telecommunications is often beyond the resources of all but a few departments.

Using the experience learned from designing, implementing, and operating a very large central site PIMS' facility has allowed the Authority to offer operational support to remote sites. This will allow the expenses associated with PIMS to remain attractive to police departments and, in addition, allows additional capabilities to be built into the system. This includes the archival of data in a readily accessible uniform format available to criminal justice decision makers and researchers.

Figure 1

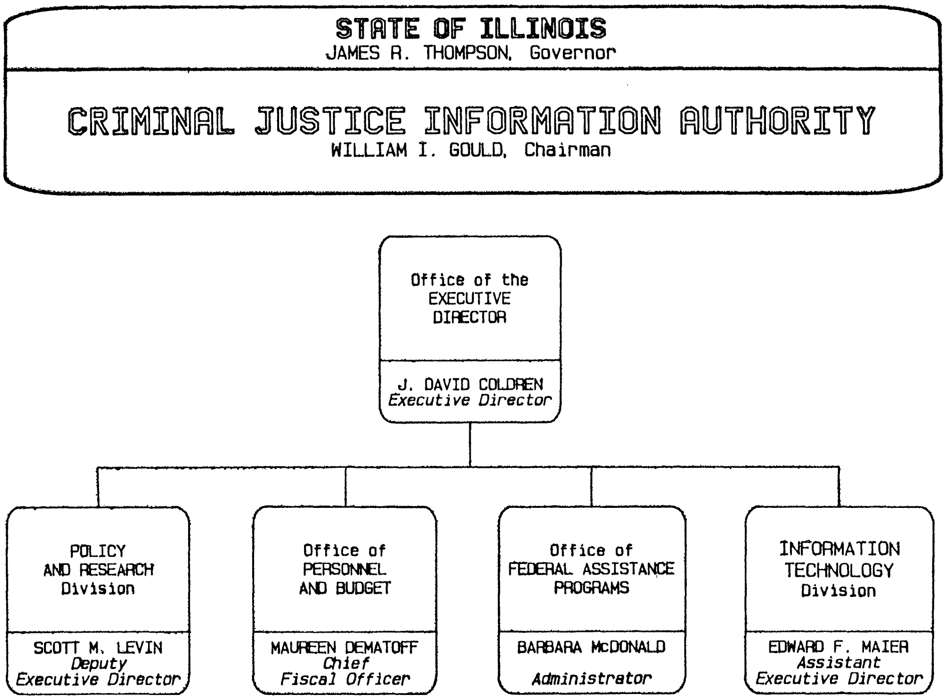
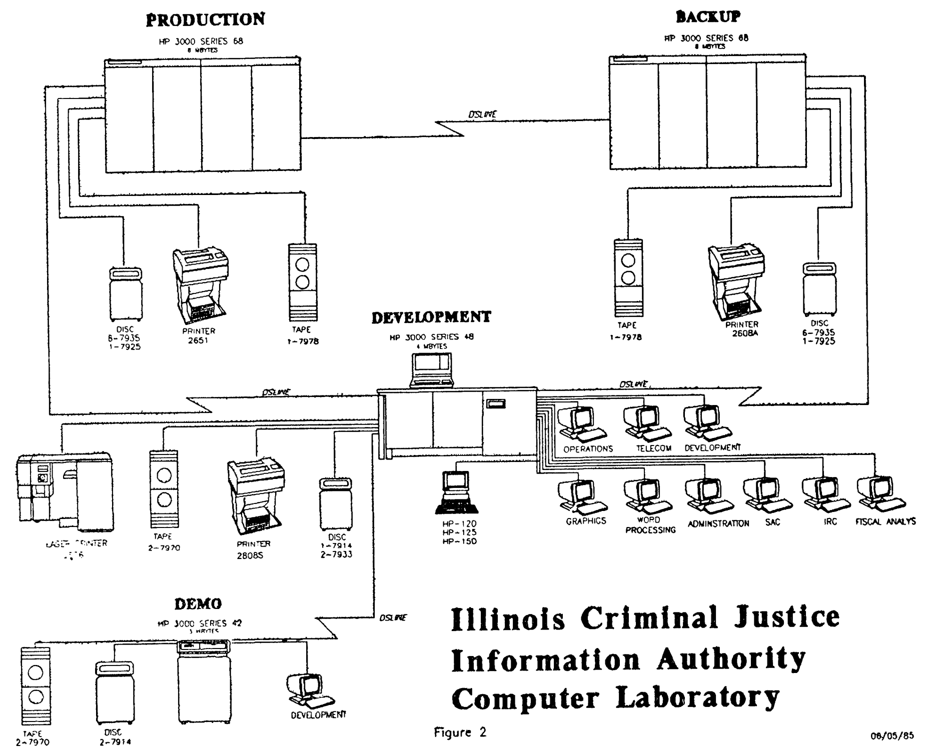

STATE OF ILLINOIS
JAMES R. THOMPSON, Governor
CRIMINAL JUSTICE INFORMATION AUTHORITY
WILLIAM I. GOULD, Chairman
Office of the EXECUTIVE DIRECTOR
J. DAVID COLDREN Executive Director
POLICY AND RESEARCH Division
SCOTT M. LEVIN Deputy Executive Director
Office of PERSONNEL AND BUDGET
MAUREEN DEMATOFF Chief Fiscal Officer
Office of FEDERAL ASSISTANCE PROGRAMS
BARBARA McDONALD Administrator
INFORMATION TECHNOLOGY Division
EDWARD F. MAIER Assistant Executive Director

Figure 2



Illinois Criminal Justice Information Authority Computer Laboratory

Figure 2

Figure 3

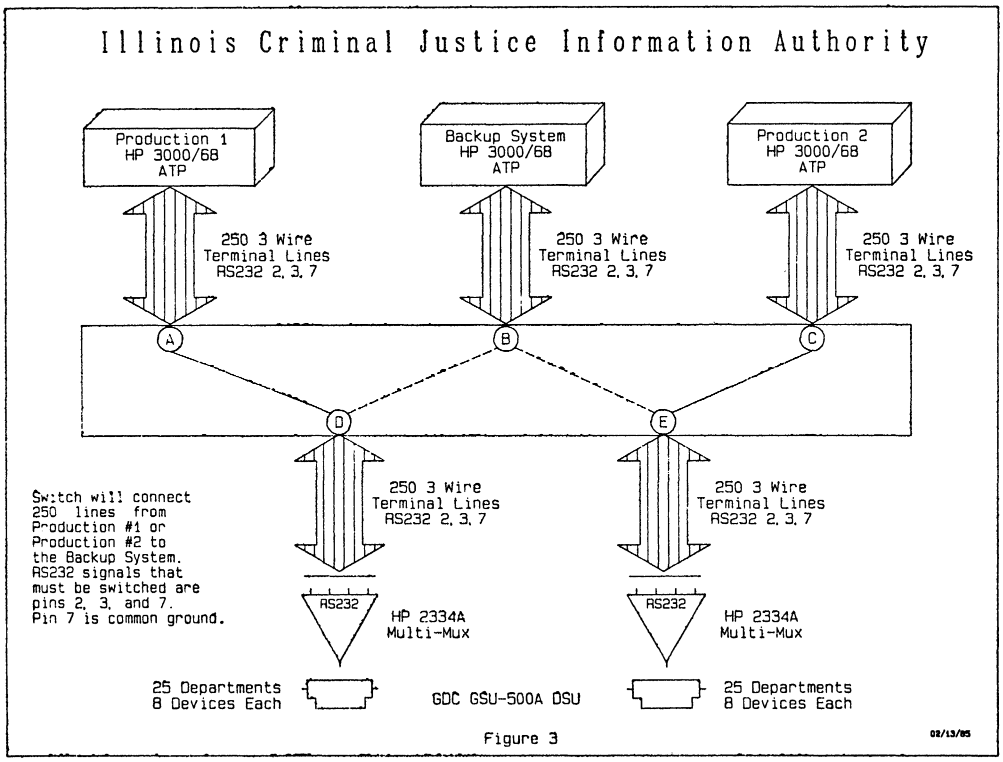
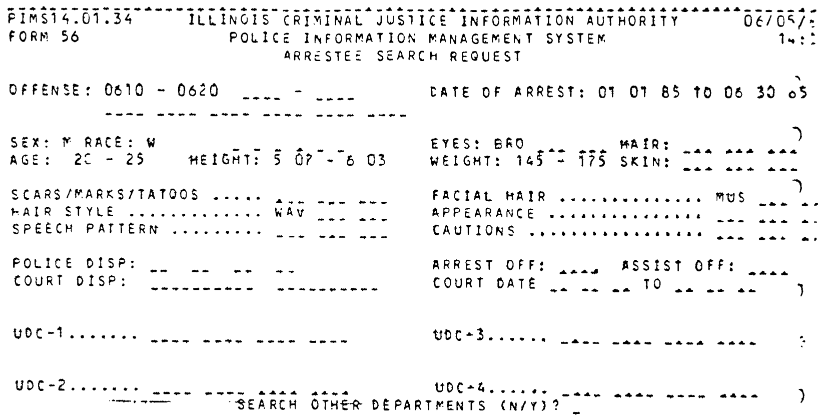
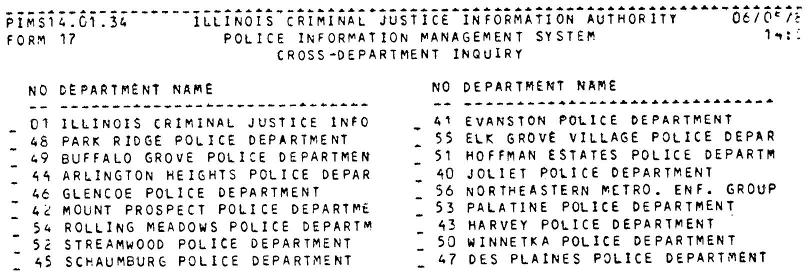

Illinois Criminal Justice Information Authority
Production 1
HP 3000/68
ATP
Backup System
HP 3000/68
ATP
Production 2
HP 3000/68
ATP
250 3 Wire
Terminal Lines
RS232 2, 3, 7
A
B
C
D
E
Switch will connect 250 lines from Production #1 or Production #2 to the Backup System. RS232 signals that must be switched are pins 2, 3, and 7. Pin 7 is common ground.
RS232
HP 2334A
Multi-Mux
25 Departments
8 Devices Each
GDC GSU-500A DSU
Figure 3
02/13/85

Figure 4



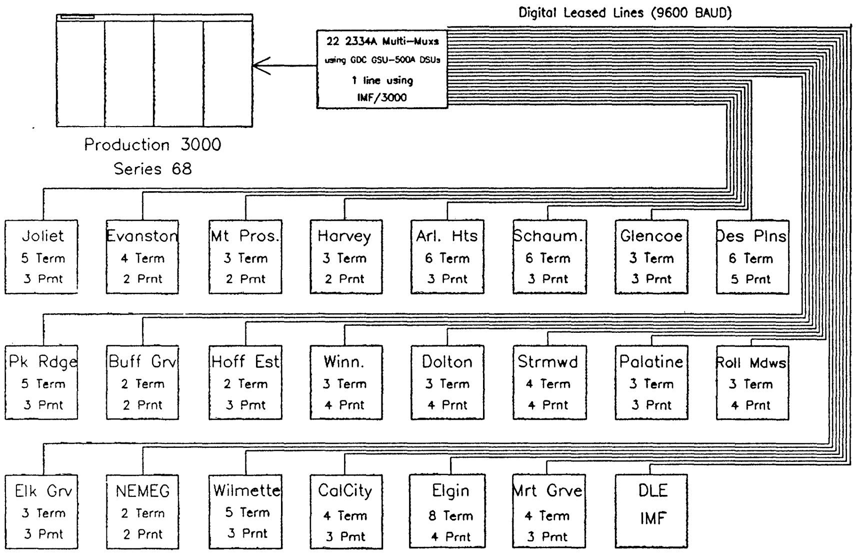
```
PIMS14.01.34      ILLINOIS CRIMINAL JUSTICE INFORMATION AUTHORITY      06/05/
FORM 56                  POLICE INFORMATION MANAGEMENT SYSTEM             14:
                               ARRESTEE SEARCH REQUEST

OFFENSE: 0610 - 0620  ____ - ____           DATE OF ARREST: 01 01 85 TO 06 30 85
           ____ ____ ____ ____ ____ ____

SEX: M RACE: W                              EYES: BRO ___ ___  HAIR: ___ ___ ___
AGE:  20 - 25    HEIGHT: 5 07 - 6 03        WEIGHT: 145 - 175 SKIN: ___ ___ ___

SCARS/MARKS/TATOOS ..... ___ ___ ___        FACIAL HAIR .............. MUS ___ _
HAIR STYLE ............. WAV ___ ___        APPEARANCE ............... ___ ___ _
SPEECH PATTERN ......... ___ ___ ___        CAUTIONS ................. ___ ___ _

POLICE DISP: __  __  __  __                 ARREST OFF: ____  ASSIST OFF: ____
COURT DISP:  __________  __________         COURT DATE __ __ __ TO __ __ __

UDC-1....... ____ ____ ____ ____            UDC-3...... ____ ____ ____ ____

UDC-2....... ____ ____ ____ ____            UDC-4...... ____ ____ ____ ____
                         SEARCH OTHER DEPARTMENTS (N/Y)? _
```



```
PIMS14.01.34      ILLINOIS CRIMINAL JUSTICE INFORMATION AUTHORITY      06/05/
FORM 17                  POLICE INFORMATION MANAGEMENT SYSTEM             14:
                              CROSS-DEPARTMENT INQUIRY

   NO DEPARTMENT NAME                        NO DEPARTMENT NAME
   -- ------------------------------         -- ------------------------------
 _ 01 ILLINOIS CRIMINAL JUSTICE INFO       _ 41 EVANSTON POLICE DEPARTMENT
 _ 48 PARK RIDGE POLICE DEPARTMENT         _ 55 ELK GROVE VILLAGE POLICE DEPAR
 _ 49 BUFFALO GROVE POLICE DEPARTMEN       _ 51 HOFFMAN ESTATES POLICE DEPARTM
 _ 44 ARLINGTON HEIGHTS POLICE DEPAR       _ 40 JOLIET POLICE DEPARTMENT
 _ 46 GLENCOE POLICE DEPARTMENT            _ 56 NORTHEASTERN METRO. ENF. GROUP
 _ 42 MOUNT PROSPECT POLICE DEPARTME       _ 53 PALATINE POLICE DEPARTMENT
 _ 54 ROLLING MEADOWS POLICE DEPARTM       _ 43 HARVEY POLICE DEPARTMENT
 _ 52 STREAMWOOD POLICE DEPARTMENT         _ 50 WINNETKA POLICE DEPARTMENT
 _ 45 SCHAUMBURG POLICE DEPARTMENT         _ 47 DES PLAINES POLICE DEPARTMENT
```

Figure 5

PIMS Configuration as of 6/85



86 Remote Terminals
68 Printing devices, including Split Screens.

Terminals are either HP2626A or HP2628A. Split screen applications are counted as a printer device, otherwise the printers are HP2334A.

06/05/85

Figure 6

```
*                   ILLINOIS  CRIMINAL  JUSTICE  INFORMATION  AUTHORITY                  *
******************************************************************************************
*                                                                                        *
*       DATE OF PROBLEM: [DATE    ]          TIME OF PROBLEM: [TIME  ]                   *
*                                                                                        *
*        OPERATOR'S INITIALS: [OI]           SOFTWARE PROBLEM: [SP]                      *
*                                                                                        *
*  WAS PROBLEM FIXED IN 1 TO 10 MINUTES ? [UT]   IN 10 TO 30 MINUTES ? [TT]               *
*                                                                                        *
******************************************************************************************
* * * * * *                                                                    * * * * * *
* * * * * *    D E S C R I P T I O N   O F   T H E   P R O B L E M             * * * * * *
* * * * * *                                                                    * * * * * *
******************************************************************************************
*                                                                                        *
*   [P1                                                                    ]             *
*   [P2                                                                    ]             *
*   [P3                                                                    ]             *
*   [P4                                                                    ]             *
*   [DT                    ]                                                             *
*                                                                                        *
******************************************************************************************
```

Figure 7

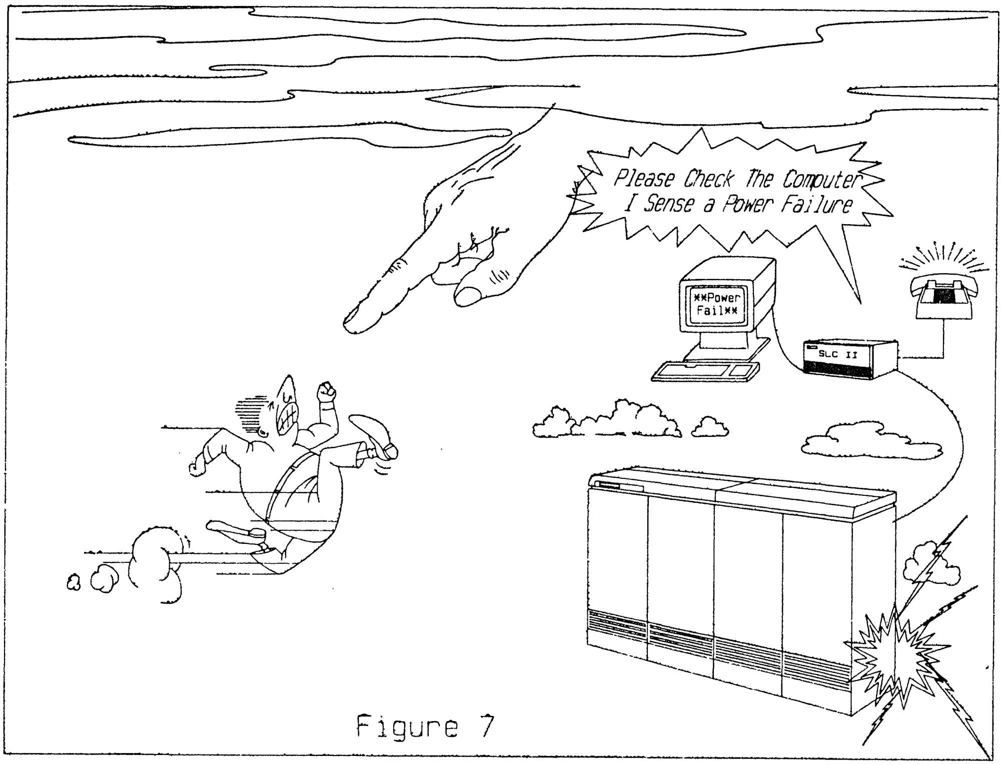


Figure 7

3081. INFORMATION SYSTEMS PROTOTYPING: A PROVEN APPROACH FOR EFFECTIVE APPLICATION DESIGN AND DEVELOPMENT

Orland Larson
Hewlett-Packard
19447 Pruneridge Ave.
Cupertino, California 95014

One of the most imaginative and successful techniques for clarifying user interfaces and generally improving the productivity and effectiveness of application development is a methodology called INFORMATION SYSTEMS PROTOTYPING.

With waiting time for new applications running into several years and those applications failing to meet the users needs, managers as well as users have been searching for more efficient and effective approaches to systems development.

Prototyping, as an application system design and development methodology, has evolved into a real option for both the MIS professional and the user.

This paper reports on the growing body of knowledge about prototyping. It begins by reviewing the changing role of data processing, the challenges facing the MIS organization, and the traditional approach to application development. It then defines prototyping followed by the step-by-step prototype development process. The advantages and disadvantages, as well as the cost and efficiency of prototyping, will be discussed followed by the essential resources neccessary to effectively prototype applications. In conclusion, to illustrate the benefits of prototyping, the speaker will present success stories of systems developed using the prototyping approach.

INTRODUCTION

The Changing Role of Data Processing

The data processing department has changed dramatically since the 1960s, when application development as well as production jobs were usually run in a batch environment with long turnaround times and out-of-date results.

The 1970s were a period of tremendous improvement for the data processing environment. One of the key developments of that period was the development and use of Data Base Management Systems (DBMS). This provided the basis for on-line, interactive applications. In addition, computers and operating systems provided programmers the capability of developing application programs on-line, while sitting at a terminal and interactively developing, compiling, and testing these applications. The end user was also provided with easy-to-use, on-line inquiry facilities to allow them to access and report on data residing in their data bases. This took some of the load off the programmers and allowed them to concentrate on more complex problems.

During the 1980s, the data base administrator and MIS manager will see increased importance and use of centralized data dictionaries or "centralized repositories of information about the corporate data resources." Simpler and more powerful report writers will be used by the end user and business professional. The programmer will see the trend towards the use of high-level, transaction processing languages, also known as fourth generation languages, to reduce the amount of code required to develop applications. Finally, the tools have been developed to effectively do application prototyping, which will provide benefits to the end user as well as the application programmer and analyst.

Throughout the 70s and 80s, information has become more accurate, reliable, and available, and the end user or business professional is becoming more actively involved in the application development process.

Challenges Facing MIS

One of the MIS manager's major problems is the shortage of EDP specialists. A recent Computerworld article predicted that by 1990 there will be 1/3 of a programmer available for each computer delivered in this country. Software costs are also increasing because people costs are going up and because of the shortage of skilled EDP specialists. The typical MIS manager is experiencing an average of two to five years of application backlog. This doesn't include the "invisible backlog," the needed applications which aren't even requested because of the current known backlog. In addition, another problem facing MIS management is the limited centralized control of information resources.

The programmer/analyst is frustrated by the changeability of users' application requirements (typically, the only thing constant in a user environment is change). A significant amount of programmers' time is spent changing and maintaining users' applications (as much as 60 to 80 percent of their time). Much of the code the programmer generates includes the same type of routines such as error checking, formatting reports, reading files, checking error conditions, data validation, etc. This can become very monotonous or counterproductive for the programmer.

The end user or business professional is frustrated by the limited access to information needed to effectively do his/her day-to-day job. This is especially true for those users who know their company has spent a great deal of money on computer resources and haven't experienced the benefits. The users' business environment is changing dynamically and they feel MIS should keep up with these changes. MIS, on the other hand, is having a difficult time keeping up with these requests for application maintenance because of the backlog of applications and the shortage of EDP specialists. Once the user has "signed off" on an application, he is expected to live with it for a while. He is frustrated when he requests what he thinks is a "simple change" and MIS takes weeks or months to make that change.

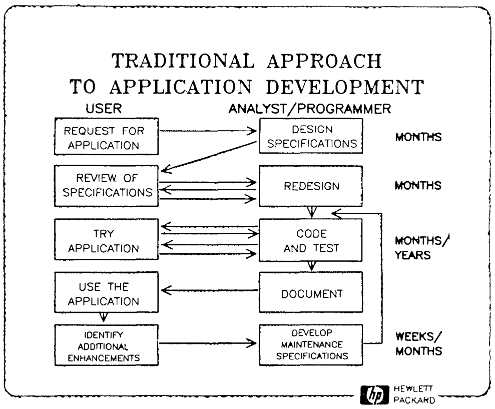
Traditional Approach to Application Development

There are some myths concerning traditional application development:

- Users know exactly what they want
- Users can effectively communicate their needs to MIS
- Users needs never change.

The traditional approach to application development has serious limitations when applied to on-line, interactive information systems that are in a state of constant change and growth. Communications among the user, analyst, programmer, and manager tend to be imprecise, a detailed analysis prolongs the process to the annoyance of the user, and specifications are either ambiguous or too voluminous to read. To compound this problem, the user is often requested to "freeze" his requirements, and subsequent attempts at change are resisted.

Let's review the traditional approach to application development.



- The user first requests an application and then an analyst or programmer is assigned to the application.

- The analyst or programmer takes the oftentimes sketchy user's specifications and designs more complete specifications.

- The user then reviews the analyst's interpretations of his specifications and probably makes additional changes.

- The analyst redesigns his specifications to adapt to these changes. (By this time, several days, weeks or months have gone by.)

- The user finally approves the specifications, and a team of analysts and programmers are assigned to develop, test and document the application.

- The user finally tries the application. Months or years may have gone by before the user gets his first look at the actual working application.

- The user, of course, will most likely want additional changes or enhancements made to the application. This is called adjusting the application to the "real world".

- Depending on the extent of these changes, additional maintenance specifications may have to be written and these program changes coded, tested and documented.

- The total application development process may take months or years, and the maintenance of these applications may go on forever.

In summary, the traditional approach to application development results in long development times, excessive time spent on maintenance, a multi-year backlog of applications, limited control and access to information, and applications that lack functionality and flexibility and are very difficult to change. The question is: "Can we afford to continue using this approach to application development?"

Prototype Defined

According to Webster's Dictionary, the term prototype has three possible meanings:

1) It is an original or model on which something is patterned: an archetype.

2) A thing that exhibits the essential features of a later type.

3) A standard or typical example.

J. David Naumann and A. Milton Jenkins in a paper on software prototyping (see reference 7) believe that all three descriptions apply to systems development. Systems are developed as patterns or archetypes and are modified or enhanced for later distribution to multiple users. "A thing that exhibits the essential features of a later type" is the most appropriate definition because such prototypes are a first attempt at a design which generally is then extended and enhanced.

Roles in the Prototyping Process

There are two roles to be filled in prototyping -- the user/designer and the systems/builder. These roles are very different from the traditional user and analyst/programmer roles under the traditional approach. The terms "user/designer" and "systems/builder" emphasize these differences and denote the functions of each participant under the prototyping methodology. Remember it is the user who is the designer of the application system and the systems professional who is the builder.

The user/designer initiates the process when he/she conceives of a problem or opportunity that may be solved or exploited by the use of an information system. The user/designer typically must be competent in his/her functional area (many times he/she is a manager) and usually has an overall perspective of the problem and can choose among alternative solutions. However, he/she requires assistance from the MIS organization.

The systems/builder is assigned by the MIS organization to work with the user/designer and is competent in the use of the available prototyping tools and knowledgeable about the organizations data resources.

Prototyping Process

The process of application prototyping is a quick and relatively inexpensive process of developing and testing an application system. It involves the user/designer and the systems/builder working closely to develop the application. It is a live, working system; it is not just an idea on paper. It performs actual work; it does not just simulate that work. It can be used to test assumptions about users' requirements, system design, or perhaps even the logic of a program.

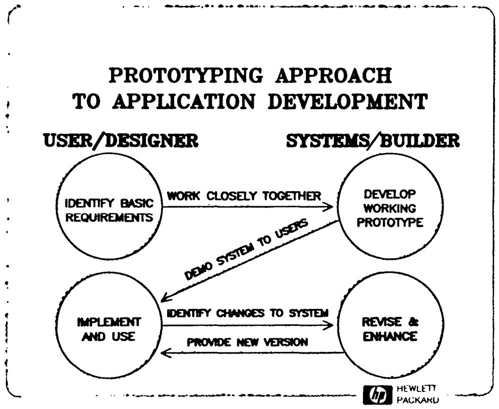
Prototyping is an iterative process. It begins with a simple prototype that performs only a few of the basic functions of a system. It is a trial and error process - build a version of the prototype, use it, evaluate it, then revise it or start over on a new version, and so on.

Each version performs more of the desired functions and in an increasingly efficient manner. It may, in fact, become the actual production system. It is a technique that minimizes the dangers of a long formal analysis and increases the likelihood of a successful implementation.

Prototyping Methodology/Model

The prototyping methodology in general, is based on the following proposition: "People can tell you what they don't like about an existing application easier than they can tell you what they think they would like in a future application."

Prototyping an information system can be viewed as a four-step procedure.



Step 1. User/designer identifies the basic information requirements:

- Write a brief, skeleton-like statement that captures the essential features of the information requirements. - User/designer and systems/builder work closely together. - Concentrate on users' most basic and essential requirements. - Define data requirements, report formats, screens, and menus. - Need not involve lengthy written specifications. - For larger systems, a design team may need to spend a few weeks preparing a first-effort requirements document.

Step 2. Systems/builder develops the initial prototype:

- Systems/builder takes the notes developed in the user discussions and quickly builds the menus and dialogs. - A data dictionary would be useful at this time. - Design and/or define data base and load subset of data. - Make use of defaults and standard report formats. - Write required application modules using a fourth generation language. - Prototype performs only the most important, identified functions.

Step 3. Users implement and use the prototype to refine requirements:

- Systems/builder demonstrates prototype to small group of users. - Users gain hands-on experience with application. - Users are encouraged to make notes of changes they would like made. - Users discuss and prioritize desired changes.

Step 4. Systems/builder revises and enhances the prototype:

- Systems/builder modifies the prototype to correct undesirable or missing features. - May require modification or redesign of data base, changes to existing programs and/or additional program modules. - Deliver back to users quickly.

NOTE: Steps 3 and 4 are repeated until the system achieves the requirements of this small group of users. Then either introduce it to a larger group of users for additional requirements or if enough users are satisfied, demo it to management to gain approval for the production system.

When to Use Prototyping

1. To clarify user requirements:

- Written specs are often incomplete, confusing, and take a static view of requirements. - It is difficult for an end user to visualize the eventual system, or to describe his/her current requirements. - It is easier to evaluate a prototype than written specifications. - Prototyping allows, even encourages, users to change their minds. - It shortens the development cycle and eliminates most design errors. - It results in less enhancement maintenance and can be used to test the effects of future changes and enhancements.

2. To verify the feasibility of design:

- The performance of the application can be determined more easily. - The prototype can be used to verify results of a production system. - The prototype can be created on a minicomputer and then that software prototype may become the specifications for that application which may be developed on a larger mainframe computer.

3. To create a final system:

- Part (or all) of the final version of the prototype may become the production version. - It is easier to make enhancements, and some parts may be recoded in another language to improve efficiency or functionality.

When Not to Use Prototyping

1. When an application requires a standard solution that already exists and is available at a reasonable cost from a software supplier.

2. When you don't have a good understanding of the tools available to prototype.

3. When the organization's data and software resources are not well organized and managed.

4. When MIS management is unwilling to develop a staff of professional systems/builders.

5. When the user/designer is unwilling to invest his/her time in the development of the application system.

Potential Problems

One of the initial problems typically encountered is the acceptance of the prototyping methodology by the systems people. This is due to the fact that people naturally tend to resist change. It may also encourage the glossing over of the systems analysis portion of a project. It is not always clear how a large complex system can be divided and then integrated. Initially, it could be difficult to plan the resources required to prototype (people, hardware and software). It may be difficult to keep the systems staff and users abreast of each version of the system. Programmers may tend to become bored after the nth iteration of the prototype. Testing may not be as thorough as desired. It might be difficult to keep documentation on the application up to date because it is so easy to change.

Even with these concerns, prototyping provides a very productive working relationship for the users and the builders. So it behooves all data processing management to learn to use this powerful tool creatively and to manage it effectively.

THE ADVANTAGES OF PROTOTYPING GREATLY OUTWEIGH THE PROBLEMS!

Advantages of Prototyping

One of the main advantages of application prototyping is that this methodology provides a capability to quickly respond to a wide variety of user requests. It provides a live, functioning system for user experimentation and accommodates changes in a dynamic user environment. One interesting aspect of this approach is that users are allowed and even encouraged to change their minds about an application's interfaces and reports, which is a very rare occurrence during the traditional approach. Maintenance is viewed right from the beginning as a continuation of the design process. Finally, prototyping provides an effective use of scarce systems/builders. One or a limited number of systems/builders will be required for each prototyping project; and while users are testing one prototype, the systems/builder can be working on another.

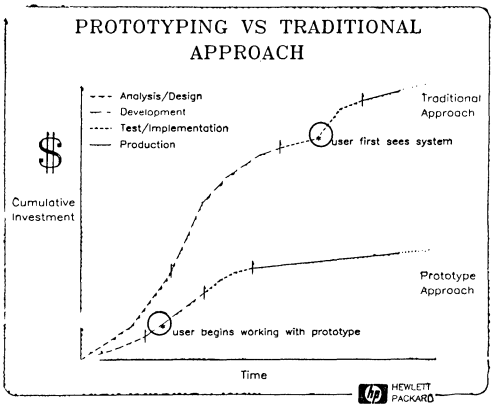
Cost and Efficiency

It has been found that there is an order of magnitude decrease in both development cost and time with the prototyping methodology.

It is often difficult to estimate the cost of prototyping an application system because the total costs of development, including maintenance, are usually lumped together. The cost of implementing the initial system is much lower than the traditional approach (typically less than 25%). However, prototyping could be expensive in the following ways:

- It requires the use of advanced hardware and software. - It requires the time of high-level users and experienced systems staff. - It requires training of the systems staff in the use of prototyping and the associated tools. - Application run-time efficiency may be compromised.

The main thing to remember is that the main focus of prototyping is not so much efficiency but EFFECTIVENESS!



Essential Resources

The following are the essential resources to effectively do application prototyping:

1. Interactive Systems (Hardware and Operating System)

   - When doing application prototyping, both the builder and the system must respond rapidly to the user's needs. Batch systems do not permit interaction and revision at a human pace. Hardware and associated operating systems tailored to on-line interactive development are ideal for software prototyping.

2. Data Management Systems

   - A Data Base Management System provides the tools for defining, creating, retrieving, manipulating, and controlling the information resources. Prototyping without a DBMS is inconceivable!

   - A Data Dictionary provides standardization of data and file locations and definitions, a cross reference of application programs, and a built-in documentation capability. These are essential to managing the corporate resources and extremely useful when prototyping.

3. Generalized Input and Output Software

   - Easy to use data entry, data editing, and screen formatting software are extremely helpful in the application prototyping process to allow the programmer to sit down at a terminal with a user and interactively create the user's screens or menus.

   - Powerful, easy-to-use report writer and query languages provide a quick and effective way of retrieving and reporting on data in the system. A report writer that uses default formats from very brief specifications is most useful in the initial prototype.

   - A powerful graphics capability can be extremely useful for the display of data in a more meaningful graphical format.

4. Very High Level (Fourth Generation) Languages

   - Traditional application development languages such as COBOL may not be well suited for software prototyping because of the amount of code that has to be written before the user sees any results.

   - Very powerful fourth generation languages that interface directly to a data dictionary for their data definitions are ideal. One statement in this high level language could realistically replace 20-50 COBOL statements. This reduces the amount of code a programmer has to write and maintain and speeds up the development process.

5. Documentation Aids

   - Tools to aid in the maintenance of programs written in a 4GL.

   - Tools to aid in maintaining user documentation on-line.

6. Libraries of Reuseable Code

   - A library of reusable code to reduce the amount of redundant code a programmer has to write is an important prototyping resource.

   - This code could represent commonly used routines made available to programmers.

Hewlett-Packard's Tools for Prototyping

Hewlett-Packard is one of the few vendors that supplies the majority of the tools needed to effectively do software prototyping.

* Interactive Systems

  - HP 3000 (All Series) - MPE Operating System

* Data Management Systems

  - IMAGE/3000 - KSAM/3000 - MPE files - DICTIONARY/3000

* Generalized Input/Output Software

  - VPLUS/3000 - QUERY/3000 - REPORT/3000 - INFORM/3000 - HPEASYCHART - DSG/3000

* Very High Level Languages

  - TRANSACT/3000

* Documentation Aids

  - HPSLATE - HPWORD - TDP/3000

Note: There are several additional excellent prototyping tools available from HP third-party vendors which are too numerous to mention here. Please consult the Hewlett-Packard Business

Systems Software Catalog (Part No. 3000-90251) for more information.

Summary

Information Systems Prototyping is truly a "state-of-the-art" way of developing on-line interactive applications.

- Prototyping promotes an interactive dialogue between the users and the programmer, which results in a system being developed more quickly, and results in an interactive development approach which is friendlier for the end user.

- The prototype provides a live working system for the users to experiment with instead of looking at lengthy specifications.

- The users are provided with an early visualization of the system which allows them to immediately use it.

- The users are allowed and even encouraged to change their minds about user interfaces and reports.

- Maintenance is viewed right from the beginning as a continuous process and because the prototype is usually written in a very high-level language, changes are faster to locate and easier to make.

- Information systems prototyping results in:

  * Users who are much more satisfied and involved in the development process.

  * Systems that meet the user's requirements and are much more effective and useful.

  * Improved productivity for all those involved in software prototyping: the user/designers and the systems/builders.

Biography

Orland Larson is currently Information Resource Management Specialist for Hewlett-Packard. As the data base and application development specialist for the Information Systems Group he develops and presents seminars worldwide on data base management, application prototyping and productivity tools for information resource management. He is a regular speaker at Hewlett-Packard's Productivity Shows and also participates in various National Data Base and 4th Generation Language Symposiums. His experience includes the development of a methodology for designing data bases and the application of software tools to measure data base performance. Previously he was the Product Manager for IMAGE/3000, Hewlett-Packard's award winning data base management system.

Before joining HP he worked as a Senior Analyst in the MIS Department of a large California-based insurance company and prior to that as a Programmer/Analyst for various software companies. Mr. Larson has been with Hewlett-Packard since 1972.

Bibliography

Boar, Bernard H., Application Prototyping: A Requirements Definition For The 80's, John Wiley & Sons, New York, New York, 1984.

Canning, Richard G., "Developing Systems By Prototyping," EDP Analyzer (19:9) Canning Publications, Inc., September 1981.

Jenkins, A. Milton, "Prototyping: A Methodology For The Design and Development of Application Systems," Division of Research, School of Business, Indiana University Discussion Paper #227, April 1983, (41 pages).

Jenkins, A. Milton and Lauer, W. Thomas, "An Annotated Bibliography on Prototyping," Division of Research, School of Business, Indiana Discussion Paper #228, April 1983, (25 pages).

Larson, Orland J.,"Software Prototyping - Today's Approach to Application Systems Design and Development," Proceedings 1984 International Meeting HP 3000 IUG, Anaheim, California, February 26 - March 2.

Martin, James, Application Development Without Programmers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Naumann, Justus D. and Jenkins, A. Milton, "Prototyping: The New Paradigm for Systems Development," MIS Quarterly, Vol. 6, No. 3, September 1982.

Naumann, Justus D., and Galletta, Dennis F., "Annotated Bibliography of Prototyping for Information Systems Development," MIS Research Center Working Paper (MISRC-WP-82-12), September 1982.

Note: The above working paper as well as the paper by Naumann and Jenkins entitled "Prototyping: The New Paradigm for Systems Development," MIS Research Center-Working Paper (MISRC-WP-82-03), October 1981, are available for $3.00 each from :

University of Minnesota Systems Research Center
School of Management 269 19th Avenue South
University of Minnesota Minneapolis, Minnesota 55455

or by calling 612-373-7822.

Podolsky, Joseph L., "Horace Builds a Cycle," Datamation, November 1977, pp.162-186.

Wetherbe, James C., "Systems Development: Heuristic or Prototyping," Computerworld, Vol. 16, No. 7, April 26, 1982.

3082. The drama behind the System Status Bulletin (SSB)

F. Alfredo Rego

Adager
Apartado 248
Antigua
Guatemala

Operating systems control every bit in our computers. Compilers translate our programs. Editors change our files. Device drivers send bunches of bits back and forth to printers, plotters, terminals, and so on. Application systems orchestrate all kinds of resources and resource managers in an effort to carry out staggering amounts of tasks.

It is really a miracle that anything works at all! Yet, somehow, it does. But we have to keep an eye open for ANYTHING that may have an effect (favorable or otherwise) on our careful delegations of tasks.

One way (which I follow religiously) is to read, every two weeks, the System Status Bulletin (SSB). Strange but necessary stuff, indeed.

My wife wonders why, when I read the SSB, I act as if I was reading a gripping novel or attending a melodrama that makes me go through a wide range of feelings. To share with her the reasons behind my emotional outbursts, I decided to go into the equivalent of theatrical productions using some of the hidden plots in the SSB as my inspiration.

In this article, I give a full dramatic expansion for Known-Problem Report (KPR) number 5000035840 and the plots for several other KPR's. Please note that the "plots" are just textual quotations from the System Status Bulletin.

For a live presentation, please come to the session itself at the Washington Conference. There are a few "surprise" dramas, with door prizes for those who discover the solutions for some cute mysteries!

KPR #: 5000035840 Product: MPE V/E

Keywords: FILESYS

One-line description:

Last 2 words of FILE LABEL (device class) are being overwritten with %1

Cause:

In procedure FOPENDA in the file system, the file label is read in, updated and then written out whenever an old file is opened. A buffer is allocated on the stack which the file label is read into. A delete statement exists in the code, in between the time the file label is read and then written, which deletes the top word of the stack (the last word in the file label -- the last two characters in the device class). Therefore, the last two characters in the device class in the file label are wrong anytime a file is opened as an old file.

Fix information:

Fixed in a future release of MPE.

Here are my dramatic notes:

Something very mysterious happens to the one-line description as it moves from left stage to right stage. The first phrase, which reads "last 2 words" on the left, becomes "%1" as the last expression on the right. The solution is hidden in the parenthetical expression in the "Cause" section, which explains that the last 1 (one) word is actually equivalent to the last 2 (two) characters in the device-class name field of the file label. This dramatic technique keeps the audience on edge and is very simple to use: Just have your characters mix their words.

Even though I bought a new magnifying glass just for the occasion, I could not find WHERE in the code that delete statement exists. Hopefully, somebody knows. The "Fix information" section does not commit anything toward the solution of the mystery: "Fixed in a future release of MPE." Which release? When? Is there a patch available?

Fortunately, this particular plot is just moot, deprived of practical significance, and of interest only to obscure academicians, as confirmed by the sentence "Therefore, the last two characters of the device class in the file label are wrong anytime a file is opened as an old file."

As the hero's next-to-last words, we should have these anguished questions at the same time that the stage fills with orange smoke:

- How can I design a system that NEVER opens old files?

- What happens when my system creates a file in device-class SysDisc (for instance) with 32 extents but only 1 extent allocated? After I fill up the first extent and the file system goes about its business and grabs the device-class name from the file label (finding SysDis instead of SysDisc), will it default to DISC or will it default to a system failure?

Since this is a participation drama, the LAST words of the hero are left as an exercise for the reader!

KPR #: 9999033318 Product: HPEASYCHART

One-line description:
CONTROL Y will sometimes cause pen to drag on plotter

Problem:
Using CONTROL Y while a chart is plotted, sometimes causes the pen to drag intermittently before the plotting stops.

Temporary solution:
Use Control-Y in plotting at your own risk.

KPR #: 4700144915 Product: MPE V/P

One-line description:
STORE does not report that a disk file is corrupted.

Problem:
STORE didn't detect that a disc file was corrupted and still wrote it to the tape. RESTORE, however, reported catastrophic error.

KPR #: 5000060418 Product: MPE V/E & T-MIT

One-line description:
SYSDUMP to cartridge tape aborts because of stack overflow

Problem:
Sysdump to the cartridge tape in a batch job was aborted because of stack overflow.

Cause:
The cartridge tape buffer takes too much stack space which caused stack overflow.

Temporary solution:
When dumping users' files, use STORE instead of SYSDUMP because SYSDUMP uses some additional stack space.

KPR #: 5000015610 Product: MPE

Keywords: SYSDUMP

One-line description:
doing reload, user gets "short file not restored", losing data

----

KPR #: 4700048454 Product: MPE

Keywords: SF10

One-line description:
SF10 results from HIOMDSC2 SXIT %0 with invalid TOS

Problem:
When a media error is encountered during a verify option to a CS80 disc, the disc driver will inadvertently branch to a random code location.

Fix information:
Fixed in S-MIT (MPE V)

KPR #: 4700051243 Product: MPE

Keywords: IPC SYSHALT

One-line description:
Memory header and trailer corruption, application uses IPC & process handling.

Problem:
System halt 4 caused by accessing IPC files in copy mode.

Cause:
When closing Message file opened in copy mode the system assumes the file is buffered. The file is closed and data is moved from a non-existent buffer to the file. Causing random system failures. Memory may be corrupted causing region headers and trailers to be destroyed.

Fix information:
Fixed in future releases of MPE.

KPR #: 5000044479 Product: MPE V/E

Keywords: RESTORE

One-line description:
RESTORE causes DFS corruption and a CNTL A prompt on the console.

Cause:
This problem is caused by attempting to RESTORE a file for which a

copy currently exists on the system, and has a single extent, and the file labels extent size and last extent size are different values. In this case, RESTORE will deallocate more space than the file currently owns.

Temporary solution:
Purge the old copies of files with these characteristics prior to RESTORing.

Signed off 01/29/85 in release G01.00

KPR #: 4700119974 Product: MPE V/P

Keywords: TAPE

One-line description:
If 7974 cannot write an ID due to retries, MPE will loop and reject tape.

Problem:
When STORE encounters a bad tape it attempts to rewind and unload the tape using FCONTROL. FCONTROL will write a file mark prior to the rewind unload. If the file mark write fails then the rewind unload is aborted. As a result the tape remains on line and STORE assumes that a new reel has been mounted.

Fix information:
The fix is to check the status of the FCONTROL. If it fails then force a rewind using ATTACHIO if not a remote file.

The problem was fixed in MPE V/E.

KPR #: 5000030676 Product: MPE V/P

Keywords: SF672

One-line description:
IPC problems

Problem:
Various system failures can occur when using IPC files that have been open during a previous system failure. These include SF672, SF59, and SF495.

Cause:
The principle behind IPC is that process to process communication should be quick. IPC avoids as many disc reads/writes as possible to help keep the process to process transfer time to a minimum. The side effect from this strategy is that the file label on the disc may not reflect the actual state of the "FILE" (whose records may

have never been written to disc). To solve this problem, additional checks were added when the file is first FOPENed.

Temporary solution:
Always purge an IPC file and rebuild it before using.

Fix information:
Fixed in E.00.01. Patch available.

KPR #: 4700014530 Product: MPE

Keywords: SF59

One-line description:
SF59 on Q-MIT when accessing a file

Problem:
SF59 on Q-MIT when accessing files. It could happen when the file is opened, closed, or whenever the FCB of the file is accessed.

Cause:
The problem occurs because the ACB share count can be damaged due to the fact that the ACB does not remain locked in FCLOSE when decrementing the ACB share counts (see SR#4700-38562). This causes the FCB to be deleted upon the last FCLOSE of the file but the ACB has not been deleted. Then, when the file is re-opened, the FCB vector in the ACB is reused. However, this FCB vector is not valid or points to another file's FCB and havoc results.

KPR #: 1600019042 Product: HPSPELL

Keywords: ERRORMESSAGE

One-line description:
error messages spelled incorrectly

With this cheerful and delightful ERRORMESSAGE, we end our notes.

=========
Biography
=========

F. Alfredo Rego is Adager's Research & Development Manager. He has worked with Hewlett-Packard instruments since 1966, when he was a Physics research assistant in the Center for Nuclear Studies at The University of Texas. In the 1970's he worked as a university professor in Guatemala, teaching courses in Theoretical Mathematics, Physics and Computer Science. He has worked exclusively with IMAGE databases and Adager (The Adapter/Manager for IMAGE/3000 Databases) since 1978.

The HP3000 International Users Group honored him with the 1980 Hall of Fame Award, which reads: "Outstanding Contributor, for exemplary service to the Group and its membership".

3083. At the Information Crossroads of Operating Systems: UNIX* thru the Eyes of MPE

Sam Boles, Member Technical Staff
Hewlett-Packard

With UNIXA evolving as the de facto "industry standard" operating system, Hewlett-Packard now includes this important dimension in its array of computer technology. The Series 200 and 500 of the HP9000 family currently support HP-UX, a powerful dialect of UNIXA, with more under development.

In engineering productivity ideas at the HP Productivity Center in Cupertino, California, a key element is blending the HP3000 MPE environment with the HP9000 HP-UX environment. In the light of this experience you can get a view of UNIXA thru the eyes of MPE. The friendly vernacular of MPE -- second language to HP3000 users around the world -- becomes the familiar basis in terms of which those new to UNIXA can acquaint themselves with the terse power of this operating system.

Starting with fundamentals that map one-to-one, you'll see some MPE UDC's used with HP summer students to accelerate their productivity by providing a transitional mechanism from their UNIXA background. From there you'll move into the more complex facilities that the UNIXA productivity engine gives to both programmers and end-users alike.

. . . the Hewlett-Packard

Productivity Center:

getting Performance

to the Bottom

Line . . .

At the Hewlett-Packard Productivity Center in Cupertino, California, we deal with a wide range of hardware and software products. This is part of getting the right tool to do the right job and to interact with the other tools in the right way. That's our job: Productivity: Getting Performance to the Bottom Line.

In the course of doing this, we integrate into our Productivity Network a wide range of Hewlett-Packard computers. A component of this task is to make the HP3000 and the HP9000 play together. This means that the engineers working in the Productivity Center need to be multi-lingual: conversant in MPE, UNIXA and other operating systems.

Much of our HP9000 work at the Productivity Center uses the HP-UX operating system. Both the 16-bit Series 200 and the 32-bit Series 500 support HP-UX. HP-UX is a powerful UNIXA dialect that is based on the Bell Labs System III, enhanced with features from the Berkeley 4.1 and the Bell System V, plus some unique Hewlett-Packard contributions.

The Productivity Center leverages the SEED (Student Employment and Educational Development) program that Hewlett-Packard has sponsored for many years. The SEED program is designed for the mutual benefit of Hewlett-Packard and outstanding students in universities around the world.

*UNIXA is a trademark of AT&T Bell Laboratories. The student gains the benefit of experiencing a real-world job doing real-world work on a real-world project. The company gains the benefit of bringing bright fresh creativity to its product development from youngsters who've not yet learned that a given task is impossible. Since they don't have the experience to know that it's impossible they sometimes actually get it done. Or come up with a work-around we hadn't thought of before.

Sure, we throw a lot of stuff away. But once in a while we get some super products. Probably at about the standard R&D hit ratio.

. . . MPE UDC's

to leverage the

SEED student's

knowledge of

UNIXA . . .

Coming from the university community our SEED students typically have UNIXA experience. UNIXA was born in the quasi-university environment of Bell Labs, and some of its great evolutionary contributions have come from universities such as the University of California at Berkeley.

In order to leverage this knowledge and to expedite SEED productivity in the MPE environment, we have some MPE UDC's (User Defined Commands) that map fairly closely to what the SEED student is accustomed to under UNIXA. This provides a transitional mechanism that enables the SEED student to focus more on the meat of the project than on some of the accidental properties of the operating environment.

This same transitional vehicle, if we shift it into reverse gear, can give an experienced MPE user a glimpse of UNIXA. That's what we do in this paper.

The initial emphasis is on the functional similarities of the two operating systems. However you can't deal with the similarities for very long without touching on the differences. Here you get a superficial introduction to some of the powerful UNIXA features like piping, redirection and the UNIXA file system; and, to give equal time, some of the MPE features missing from some of the popular shells (command interpreters), such as the MPE REDO (extensively used by some of us heavy-fingered folks whose data-processing lives tend to be as redundant as they are heavy-fingered.)

. . . since 1969 . . .

a little UNIXA

lore, mystique

and

culture . . .

Before we get into the UDC's let's look at a little UNIXA lore, mystique and culture. Anything that's survived in this business since 1969 (for the children among us, that's the year IBM "unbundled" in recognition of the fact that software was no longer just the packing that came free with the hardware to keep it from rattling around in the carton) has some lore and deserves a little nostalgia.

First of all, it's an operating system of the programmers, for the programmers, by the programmers. That's why we byte hacks love it. And that's why civilized people spend large sums of money for commercial shells to cover the lean terse power of UNIXA.

Some say it's unfriendly. But you have to remember that one man's "friendly" may be another man's "verbose." It's the classic issue of efficiency vs friendliness. Do you have an elaborate high-overhead ritual to establish friendliness or do you get right to the point? Do you ask for a confirmation that the user really wants to purge every file in his group, or do you assume that if he's smart enough to ask for it you'll do it for him?

You can look at it this way: like a lot of development nodes, in my section in Cupertino, we save trees by running OUTFENCE high, going to hardcopy on only a small portion of our output. That means to get hardcopy you need to

```
ALTSPOOLFILE #Onnn;PRI=nn
```

Now that's mnemonic and intuitive, probably. It's maybe what you'd call friendly. But about the third time you do it, you decide it's worth the trouble of updating your UDC's with something like

```
ASF SPL=0,PRI=1,COPIES=1
OPTION LIST
ALTSPOOLFILE #O!SPL;PRI=!PRI;COPIES=!COPIES
```

that you invoke with

```
ASF nnn nn
```

That's the UNIXA style. It's terse. All right, cryptic. Powerful. A rich repertoire of commands and options to do the kinds of things programmers do to build and document software.

. . . maybe something

of a misnomer:

today, there's little

UNI in UNIXA beyond

UNIfying . . .

Next, UNIXA may be a misnomer. Legend has it that when the brilliant Ken Thompson named his brilliant child, he did it in counterpoint to the multi-tasking multi-user MULTICS at MIT. His was a single-user system for a single-engineer work station. Today there is little UNI in UNIXA beyond the fact that it may be the single most UNIfying element across the wide variety of hardware architectures and configurations in the industry today. Beyond that great UNIfying attribute and signal contribution, UNIXA is MULTItasking, MULTI-user, MULTI-noded, MULTI-shelled, even MULTI-processed on the HP9000/500 with its tri-CPU design.

One aspect of the "non-UNI" of UNIXA is its multiplicity of dialects. This is probably good and bad at the same time. Like any worthwhile software system, UNIXA is evolving. It's inevitable that such a magnificent theme have myriad variations. The Bell Labs UNIXA, perhaps the seat of orthodoxy, has a System III and a System V. There's the Berkeley 4.1. And HP-UX, XENIX, VENIX, QNX, UNI-plus, -star, -plex and, of course, the powerful NIX of the anti-UNIXA clingons.

Now no one disputes the value to mankind of the c-shell re-do (painfully missing in some Shells), but when you transition from Brand X to Brand Y, is it there or isn't it? Not a big deal. But neither is Life, for that matter. Just a fabric (or hodge-podge) of little deals -- but if they gang up on you, you can be in trouble.

Now we can't hold back tomorrow. We don't even want to. Just don't be deluded into thinking that the UNIXA "industry standard" is going to get you entirely out of the technological retread business we've been confronted with for generations (computer, that is), with all its learning curve entropy and proactive inhibitions.

As Churchill once said about Democracy: It's not perfect by any means; it's just the best we've been able to come up with. The same assessment might apply to UNIXA.

Of course, you could see it coming. Back in ancient times, on the 1401, you could put a complete program on a single 80-column card to read-and-list cards. It started out

```
,008015 . . .
```

The comma set a word-mark to give you a "variable word-length" machine. You'd slap that one card on the front of your "source deck" to do a

```
1,$p
```

(That's UNIXA editor for "/LIST ALL.") When the 360 arrived it had a thing called an "operating system" and you couldn't do that with just one card anymore. The beginning of the end.

So much for yesterday. Let's take a look at tomorrow.

. . . UNIXA and MPE

side by side,

going thru a few

ordinary everyday

commands . . .

First a few words about the examples you see here. We all know the frustration of the example that doesn't work. One way to reduce that problem is to capture the example right at the screen in vivo.

Now to do this for the MPE part is a fairly straightforward exercise if you have an old 2647 like mine. You work the example on the terminal on-line to the computer, then position the cursor at the start of the example. You go into local command mode to do a

```
   COPY ALL FROM DISPLAY TO LEFT TAPE
```

When you've gotten the latest batch of examples on tape, you do a

```
   MARK LEFT TAPE

   REWIND LEFT TAPE
```

Then you get into TDP (Text & Document Processor), find the place where you want to splice in the example, do an

```
   /A nnn.nn
```

Then when you get the line number prompt, touch the READ key to get the cartridge tape contents spliced into your text file.

Getting the UNIXA examples is a little different. If you're using an HP9000/520, you've got an integrated 5" floppy disc drive. As you do the examples you precede the example with an echo or a cat >> to the disc file where you're collecting your actual examples. For example, for the ps (process show) command:

```
   echo $ ps -e >> seb
```

(The >> means to append or concatenate the string after echo to the target file seb.) Then you actually do the command but redirect the output to the same file:

```
   ps -e >> seb
```

This gives you what would have been on the screen in your disc file. Then, if you haven't bothered to engineer any better datacom, you can

```
   lifcp seb /dev/rfd:SEB
```

to get your examples onto a floppy in LIF (Logical Interchange Format), take it over to your HP9000/236, do a virtual terminal file transfer to the HP3000 where the main body of your text is for doing your laser typesetting via TDP, and join the examples into the appropriate spot.

In the examples you see the HP-UX form, then the MPE form with an HP-UX-like UDC. In the UDC there's an option list to show you how the UDC gets expanded and executed. Imagine a Carriage/Cursor Return at the end of each line unless specified otherwise. If there's a Control-D you'll see [ctl-D].

So much for the logistics. Let's get started. First, get on the system:

HP-UX:

```
login: boles

Welcome to Hewlett-Packard System 9000 HP-UX
```

MPE:

```
:hello boles.cad
HP3000 / MPE V  G.B0.00 (BASE G.B0.00).  MON, DEC 24, 1984,  4:24 PM
```

Accounting in HP-UX is done generally at the user level, as opposed to user.account in MPE. There are some other differences, too. For example, if you have a password and key it wrong, MPE asks you several times to try again; HP-UX doesn't tell you whether it's the user or the password or a backspace that's the problem, but asks for everything again.

HP-UX:

```
$ who am i
boles    console Dec 24 13:26
```

MPE:

```
:whoami
SHOWME
USER: #S85,BOLES.CAD,UNIX       (NOT IN BREAK)
MPE VERSION: HP32033G.B0.00.  (BASE G.B0.00).
CURRENT: MON, DEC 24, 1984,  4:26 PM
LOGON:   MON, DEC 24, 1984,  4:24 PM
CPU SECONDS: 5         CONNECT MINUTES: 2
$STDIN LDEV: 22         $STDLIST LDEV: 22
```

Notice the blanks are suppressed in the UDC to get the showme.

HP-UX:

```
$ date
Mon Dec 24 13:48:48 PST 1984
```

MPE:

```
:date
SHOWTIME
MON, DEC 24, 1984,  4:26 PM
```

Basically the same but a little more time granularity in HP-UX and a GMT (Greenwich Mean Time) basis. HP-UX:

```
$ ps -e
   PID TTY  TIME COMMAND
 27335  co  0:00 ps
 27279  co  0:04 sh
    35   ?  0:01 getty
    34  a2  0:02 getty
    33  a1  0:01 getty
    32  a0  0:01 getty
     1   ?  0:01 init
```

MPE:

```
:pse
SHOWJOB

JOBNUM  STATE IPRI JIN  JLIST    INTRODUCED  JOB NAME

#S69    EXEC        20  20       FRI  8:54A  OPERATOR.SYS
#S85    EXEC        22  22       MON  4:24P  BOLES.CAD

2 JOBS:
    0 INTRO
    0 WAIT; INCL 0 DEFERRED
    2 EXEC; INCL 2 SESSIONS
    0 SUSP
JOBFENCE= 0; JLIMIT= 5; SLIMIT= 60
```

The ps, like showjob, tells you what's running in the system. Some differences are cosmetic: syntax, format, nomenclature; but CPU consumption, state and start time are all useful but not available in both systems with these comparable commands.

HP-UX:

```
$ ls
seb
sebb
```

MPE:

```
:ls
LISTF @

FILENAME

SEBUNX
```

Again, mostly cosmetic differences.

HP-UX:

```
$ ll
total 3
-rw-rw-rw-   1 boles    101          370 Dec 24 13:51 seb
-rw-rw-rw-   1 boles    101          100 Dec 24 13:48 sebb
-rw-rw-rw-   1 boles    101          365 Dec 24 13:51 sebc
```

MPE:

```
:ll
LISTF @,2
ACCOUNT=  CAD          GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

SEBUNX   *         72B  FA          48         48   7       16  1  1
```

The HP-UX "long" file list gives security, date and owner. The -rw-rw-rw- means it's an ordinary data file (not a directory nor a device special file), the owner of the file has read and write access but not execute permission, as do the user's group and the public in general. The listing also includes number of directory links, owner, group code, size in bytes, date and time of last modification.

This touches on a major difference: the file system. The UNIXA directory structure and file concepts are a major transitional consideration, and beyond the scope of this paper. You get glimpses here in the links information and in the mkdir and cd examples below. But remember this is only the tip -- there's a real iceberg there.

HP-UX:

```
$ cat
This is to show cat with no parms.
This is to show cat with no parms.
This is line 2 of show cat.
This is line 2 of show cat.
[ctl-D]
```

MPE:

```
:cat
FCOPY FROM=;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show cat with no parms.
This is to show cat with no parms.
This is line 2 of show cat.
This is line 2 of show cat.
 < CONTROL Y >

2 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is the cat (for concatenate) form without parameters. It's basically input from $STDIN and output to $STDLIST -- the CRT in this case.
HP-UX:

```
$ cat > file1
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
[ctl-D]
```

MPE:

```
:catt file1
FCOPY FROM=;TO=file1;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
 < CONTROL Y >

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is the concatenation of CRT input to a new or replaced file on disc. An easy way to build a file without getting into the editor -- but you give up the more powerful edits. Notice the >. That's UNIXA redirection from the default CRT to the named file. Be careful: UNIXA has high regard for your presence of mind. If it finds a file out there already by that name, it doesn't ask as MPE does whether you're sure you want to purge it (unless you've removed the write permission with a chmod) -- it just writes over the old file.

HP-UX:

```
$ cat file1
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
```

MPE:

```
:catf file1
FCOPY FROM=file1;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

Here with an implicit < redirection of input, we concatenate from the named file to the CRT.
HP-UX:

```
$ cp file1 file2
$ ll file*
-rw-rw-rw-   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          121 Dec 25 20:44 file2
$ cat file2
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
```

MPE:

```
:cp file1 file2
FCOPY FROM=file1;TO=file2;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:ll file@
LISTF file@,2
ACCOUNT=  CAD         GROUP=  UNIX
```

```
FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1              80B  FA           3       1023   1      128  1  8
FILE2              80B  FA           3       1023   1      128  1  8


:catf file2
FCOPY FROM=file2;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983


This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
EOF FOUND IN FROMFILE AFTER RECORD 2

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

This is a simple file copy with no changes as you can see from the ll and cat listings. Note the wild card * that gives you all files starting with "file".
HP-UX:

```
$ cat >> file2
This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
[ctl-D]
```

MPE:

```
:cattt file2
FILE file2,OLD;ACC=APPEND
FCOPY FROM=;TO=*file2
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
 < CONTROL Y >

3 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:catf file2
FCOPY FROM=file2;TO=
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983
```

```
This is to show the translation of
the UNIX vernacular to an MPE environment.
It's getting harder.
This is some more text to illustrate the
concatenation facility of UNIX in this game
of "Follow the Leader" with MPE.
EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
```

Here you see a concatenation with the append redirection instead of the replace. Control-D signals End of Data.

HP-UX:

```
$ cp file2 file3
$ cp file2 file3b
$ cp file2 file3c
$ ll file3*
-rw-rw-rw-   1 boles    101          247 Dec 25 20:55 file3
-rw-rw-rw-   1 boles    101          247 Dec 25 20:55 file3b
-rw-rw-rw-   1 boles    101          247 Dec 25 20:55 file3c
$ rm file3*
$ ll file3*
file3* not found
```

MPE:

```
:cp file2 file3
FCOPY FROM=file2;TO=file3;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:cp file2 file3b
FCOPY FROM=file2;TO=file3b;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:cp file2 file3c
FCOPY FROM=file2;TO=file3c;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983
```

```
EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:ll file3@
LISTF file3@,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE3             80B  FA           6       1023   1       128  1  8
FILE3B            80B  FA           6       1023   1       128  1  8
FILE3C            80B  FA           6       1023   1       128  1  8
:rm file3
PURGE file3
:rm file3b
PURGE file3b
:rm file3c
PURGE file3c
:ll file3@
LISTF file3@,2
NO FILES FOUND IN FILE-SET (CIWARN 431)
```

Note here the generic purge, representative of the UNIXA respect for the programmer's presence of mind. (Some of us who only marginally deserve that respect do a lot more back-ups under UNIXA.) You can protect yourself on sensitive files by removing write permission and thereby getting UNIXA to prompt for confirmation of purge.
HP-UX:

```
$ ll file1
-rw-rw-rw-   1 boles    101         121 Dec 24 14:01 file1
$ chmod 600 file1
-rw-------   1 boles    101         121 Dec 24 14:01 file1
```

MPE:

```
:ll file1
LISTF file1,2
ACCOUNT=  CAD         GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1             80B  FA           3       1023   1       128  1  8


:chmod600 file1
SECURE file1
```

Here's an example of changing access permissions on a file. We don't have a simple parallel in MPE. This now disallows group and public users to access the file. Note that the UNIXA granularity of control could be approximated by a combination of the secure you see here and the altgroup xxx; access= facilities in MPE.

HP-UX:

```
$ cp file2 file3
$ ll file*
-rw-------   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101          247 Dec 25 21:06 file3
$ mv file3 file4
$ ll file*
-rw-------   1 boles    101          121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101          247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101          247 Dec 25 21:06 file4
```

MPE:

```
:cp file2 file3
FCOPY FROM=file2;TO=file3;NEW
HP32212A.3.18 FILE COPIER (C) HEWLETT-PACKARD CO. 1983

EOF FOUND IN FROMFILE AFTER RECORD 5

6 RECORDS PROCESSED *** 0 ERRORS


END OF SUBSYSTEM
:ll file@
LISTF file@,2
ACCOUNT=  CAD        GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1              80B  FA           3       1023   1      128  1  8
FILE2              80B  FA           6       1023   1      128  1  8
FILE3              80B  FA           6       1023   1      128  1  8


:mv file3 file4
RENAME file3,file4
:ll file@
LISTF file@,2
ACCOUNT=  CAD        GROUP=  UNIX

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX
```

```
FILE1               80B  FA          3       1023  1      128  1  8
FILE2               80B  FA          6       1023  1      128  1  8
FILE4               80B  FA          6       1023  1      128  1  8
```

Here you see the move or rename facility in action.

HP-UX:

```
$ pwd
/users/boles
$ mkdir dir2
$ ll
total 15
drwxrwxrwx   1 boles    101             0 Dec 25 21:26 dir2
-rw-------   1 boles    101           121 Dec 24 14:01 file1
-rw-rw-rw-   1 boles    101           247 Dec 25 20:53 file2
-rw-rw-rw-   1 boles    101           247 Dec 25 21:06 file4
-rw-rw-rw-   1 boles    101          2536 Dec 25 21:28 seb
-rw-rw-rw-   1 boles    101          1055 Dec 24 14:07 sebe
-rw-rw-rw-   1 boles    101          1055 Dec 25 20:43 sebf
-rw-rw-rw-   1 boles    101          1607 Dec 25 20:54 sebg
-rw-rw-rw-   1 boles    101          2059 Dec 25 21:04 sebh
$ cd dir2
$ pwd
/users/boles/dir2
$ cp ../file1 file1dir2
$ ll
total 1
-rw-------   1 boles    101           121 Dec 25 21:35 file1dir2
$ cd
$ pwd
/users/boles
```

MPE:

Here we don't have an MPE analog closer than hello with a new group specified. In the example, while in the home directory, you make a new directory with mkdir. The directory file (initial "d" in the ll listing) now appears in its parent directory. A cd (change directory) to the subdirectory dir2 is confirmed with a pwd showing the path name up the directory chain. The cp uses a ../ to indicate the parent directory of the current working directory. A cd without an explicit directory gets us back to the home directory, which the pwd confirms. This is just a quick dip in the deep end of the pool. Don't worry about this for the brief glimpse of UNIXA you get here. But do be aware that the UNIXA file system is different from MPE.

```
$ who > file4
$ cat file4
boles    console Dec 24 13:26
$ date >> file4
$ cat file4
boles    console Dec 24 13:26
```

```
Tue Dec 25 21:42:10 PST 1984
$ ll file4
-rw-rw-rw-   1 boles    101           59 Dec 25 21:42 file4
$ wc file4
      2     11     59 file4
$ grep 'MPE' file1
the UNIX vernacular to an MPE environment.
$ ll file1 > temp1; wc file1 > temp2; cat temp1 temp2 | grep 'file'
-rw-------   1 boles    101          121 Dec 24 14:01 file1
      5     22    121 file1
```

MPE:

Don't try to map this one-for-one to MPE. You see some redirection and a new counter command, wc, to set up an illustration of piping (the | operand) and the string finder, grep. The wc counts lines, words and characters. The grep lists the lines that contain the general expression ("mess": mnemonics are merely a state of mind) search string argument. Here the cat has 2 input files that it pipes to grep which outputs the two lines containing the search string 'file'. Before wrapping up, let's scratch the surface of the UNIXA shell. First some simple shell scripts. Suppose you want your UNIXA to speak MPE. Here you see a file called listf that contains ls. At first it won't execute but the chmod fixes that.

```
$ cat > listf
ls
[ctl-D]
$ listf
listf: cannot execute
$ chmod 777 listf
$ listf
dir2    file2 . . . temp1   temp3
file1   file4 . . . temp2
```

Here you see a file called purge with a $1, the symbol for the first argument, which is the name of the file you want to purge. The chmod makes it executable.

```
$ cat > purge
rm $1
[ctl-D]
$ chmod 777 purge
$ purge temp4
```

Next you see a shell control loop that edits all the files starting with "file", using the commands in sebmod.

```
$ cat sebsh
for i in file*
  do ed $i < sebmod
  done
```

Here's what sebmod looks like. It gives the file name, lists the first record, inserts a new line, then lists the first three lines, then quits.

```
$ cat sebmod
f
1
i
Begin with Sep 1985 HP3000 IUG Conference . . .
.
1,3p
q!
```

Here's an execution:

```
$ sebsh
107
file1
This is to show the translation of
Begin with Sep 1985 HP3000 IUG Conference . . .
This is to show the translation of
the UNIX vernacular to an MPE environment.

233
file2
This is to show the translation of
Begin with Sep 1985 HP3000 IUG Conference . . .
This is to show the translation of
the UNIX vernacular to an MPE environment.

59
file4
boles    console Dec 24 13:26
Begin with Sep 1985 HP3000 IUG Conference . . .
boles    console Dec 24 13:26
Tue Dec 25 21:42:10 PST 1984
```

Epilogue . . .

There you have it: a glimpse thru the Looking Glass from MPE-land into the Land of UNIXA. You've seen our "MPENIX": some of the elementary functions in our quasi-UNIXA UDC's that we built to help our SEED students. From there you sampled some of the UNIX power that enables a computer user to reach new levels of productivity. You've seen some of the features that have enabled UNIXA to establish a good track record as the common link that lets us move with reasonable gracefulness across a substantial portion of the computer world today.

About the Author . . . .

Sam Boles is a Member Technical Staff in the Productivity Center at the Hewlett-Packard computer facility in Cupertino, California. With HP since 1976, his computer experience started back in the AUTOCODER days of the 1401/1410, migrated thru the 360/370 era, and now focuses on networking HP productivity technology. Sam received his MS at UCLA in Information Systems.

3084. Change Management: An Operations Perspective

Scott Hirsch
RCM
4 Embarcadero Center
San Francisco, California 94111

-Abstract-

When an MIS department experiences a change in management, two challenges must be met: (1) Day-to-day processing must continue and (2) Overall Company needs must be evaluated in order to judge the adequacy of existing systems and to form the basis of any long-range planning. This paper will examine the role of Operations in supporting existing production during transition and address the reasons for change and how underlying problems can be avoided.

-Operations Charter-

Operations will be responsible for acting as the user interface for the MIS (or Data Processing) Department. Its functions will include, but not be restricted to: forms handling, report distribution, system performance and capacity planning, ordering and distributing supplies, system backup and disaster recovery, service requests and bug reporting, and chargeback and billing.

-How Did It Happen?-

The first question that needs to be asked when approaching a turnaround situation is how did things get as bad as they are? (I am assuming that a complete change of management would not have occurred had everything been wonderful.) The reasons I have seen include:

- MIS has no say in decision-making process.
- Breakdown in relations between MIS and end-users.
- MIS Director/Manager unqualified for position (typically a technical staffer who survived long enough to win position by default).
- Decision-makers unwilling to commit the dollars and time necessary to do the job right.
- Management problems are treated as technical problems
- MIS staff no longer up on the latest technology and methodologies.

All of the above contribute to the demise of even the smoothest running of departments. Each point will now be examined in greater detail.

-Whos Department Is It Anyway?-

In trying to turn around an MIS Operation, you want to learn from the past and avoid repeating any mistakes. The biggest mistake you want to avoid is in the decision making process. There is definitely such a thing as a no-win situation and if you have no control over the direction of your department, such is your fate. Therefore, it is the manager's responsibility to provide an adequate case for any requests to Upper Management for hardware, software, personnel, etc. However, if your well-documented budget is shot down mercilessly by the Finance Director or Budget Committee then you might as well throw in the towel right there -- the situation is hopeless. We will assume that you are not in that unlucky -- and, alas, far too common -- situation and have established the credibility necessary for the successful turnaround of your department.

-An End-User Valentine-

Let's face it: the end-user is the MIS department's reason for being. MIS is a support function and if those people you are supposed to be supporting make unflattering remarks every time you pass by, something is fundamentally wrong. Why not quantify your suspicions with a survey? Leave it anonymous so users will tell you the truth and insist that everyone submit a response. You will probably find that users are shocked by your interest in them. Take this as a great opportunity to reestablish communications with a very nice -- but frustrated -- group of people. You need these people! They will provide priceless Public Relations for you and your department when budget time rolls around. And, after all, your success is measured to a great degree by the amount of user satisfaction. This point cannot be stressed enough!

-The User Speaks-

Since you are new to the Company, you will have a lot of questions. I have found Users to be a great source of answers to many of the most pressing Operations questions -- better, in fact, than talking to MIS staff. For example, I wondered why we were churning out so much paper -- like 10 copies of a 250 page file dump -- every week. In talking with users I discovered that it used to be that 10 people relied on that report but now circumstances have changed and half the reports sit around collecting dust. Eliminating unnecessary services can be a big first step towards providing better support and will enable you to devote more staff resources to the less easy problems -- all without having to "go to the well". You also need the User as a gauge of system performance. I am not aware of many users who will call you to say that the system is performing with lightning speed, but that doesn't mean they will call and complain when it -is- slow. As much as it hurts, you should encourage this kind of negative user feedback. When system resources are preciously finite, the machine jockeys over in devlopment had better learn how to control their compiles or the payroll people might be "slow" themselves next pay period. But seriously, until you are able to get a true picture of your current and future needs so you may upgrade, the user will be a valuable resource in managing your system resources. One final point: Once you have won over the User, you will find that your job of keeping them happy is that much simpler. Your interest in them will pay off in more realistic requests (as opposed to the inflated ones your predecessors used to receive because they never produced anything) and an even greater understanding on their part of the limitations you face in meeting their needs.

-Who's on First?-

Obviously, all attention eventually returns to the state of the HP3000 itself. Where do you begin? Try starting with an observation of the personnel around you. Are they organized? Is there anything resembling a schedule? Are responsibilities clearly defined or is everyone getting in each other's way? Expect to hear from your staff that they don't have enough time to get everything done. Now, ask them to submit to you a report detailing everything they do on a daily basis -- what it is, how long it takes and, most important, its purpose. Don't be surprised if much of what they are doing is redundant or downright unnecessary. Answers like "well, we've always done it this way" are common. You are now able to do two big favors for your staff: (1) A lot of meaningless work will be eliminated from their schedule, allowing them to once again work a "normal" work week and (2) By defining responsibilites for the tasks remaining -- assuming that eliminating some tasks will require a redistribution of others -- you will keep them from getting in each other's hair and will give them a sense of accomplishment. Not to mention the fact that you will stop hearing "I thought it was -his- responsibility".

-Examining the Patient--First Glance-

And what of the HP3000 itself? It -is- the tie that binds us users together and naturally can't be overlooked. Here you, as manager, will unfortunately need to committ many non-prime time hours playing the part of sleuth. What does it take to manage the Operation side in strictly machine terms; that is, what are the minimum requirements of your HP3000 computer for you to maintain user satisfaction? First, all applications in production should run bug free and fast. The chances of that occurring in a shop in transition are next to nil. Therefore, at minimum you will need access to all source code, file definitions, documentation (oh no!) and, last but not least, a way of putting everything together: a shop standards book. I don't care how you work it, all these pieces are necessary if you want to support your users. Much has been said about programmers being "creative" and resistant to structure. That's all well and good unless you're the guy who has to find the source code for a program that suddenly starts zeroing out fields or deleting records. Begin work immediately on establishing some sort of order to your system. Any convention will work so long as it is understood by everyone in MIS and is adhered to. You may decide to keep all JOBS in a JOB group or you may elect a naming convention (both is best). Standards serve to make your job of supporting users possible by eliminating the "needle in a haystack" approach to finding the information you need when you need it. Common sense? Certainly! But how many shops maintain such discipline?

-How secure Should My System Be?-

Security is another area that varies in importance from shop to shop but is always an issue. Everybody knows that it is easier to start out with tight security and relax it over time rather than vice versa. In MPE, the one thing to remember is that Write access is equivalent to Purge access. From there you will need to decide how you want to handle passwords and lockwords. I have found general use of lockwords to be more of a nuisance than a help. Getting back to good communication, internal security problems can be avoided through good communication. Have you found one of the programmers raising and lowering the Job Fence from his terminal after running God? Rather than arbitrarily slapping on a lockword, try talking with him. If the problem persists, your problem goes beyond being technical. In this world of hackers and such, rotating passwords on a regular basis makes sense and takes little effort to implement. Obviously, password information should be kept in a secure place on the system. Just make sure that all users are kept up to date on the latest passwords. There is nothing worse than getting a wake-up call from an irate user who can't log on to the system.

-Where Did All My Disc Space Go?-

The typical approach to disc space seems to be "Since we have all these drives, let's keep as much on-line as we can". I hope that is not your situation. Lack of free space is a major contributor to poor system performance. What is less obvious is that when you actually need to find something, a LISTF can take forever. Do yourself a favor and determine as quickly as possible what must stay and what can be archived. In the worst case, you will archive a bit too aggressively and will have to restore some files. Archiving 5,000 files and then restoring 6 seems like a good deal to me.

-And Speaking of Tape-

Most HP3000 shops do not understand tape handling. Since you can't have generations of files on the system at any given time and, until recently, couldn't RESTORE a file if the creator was not in the directory, tape handling tends to be approached with some trepidation. Once again, the problem is management, not technical. Anyone can do a STORE. Not everyone can find those STOREd files or even that tape once the store is done. I assume backup is a given. H-P teaches you how to backup up and even provides sample jobs. With T-MIT, absolutely anyone can do a backup. Please remember to send your backup offsite or you are wasting your time. My main concern in this area is with being able to quickly locate the right tape with the right files on it. There are any number of simple Tape Library Management Systems that can be bought. We utilize a very simple Contributed Software Library System that has been jazzed up for our needs. You may want to take a hard look at all the hardcopy listings you have of files dumped. Remember that utilities like TAPEDIR or STAN allow you to generate hardcopy listings -- superior to STORE/SYSDUMP in my opinion -- when needed. If you are running out of file space for your dump listings, this may provide a suitable compromise to keeping hard copies of everything.

-You're Breaking My Batch!-

Of all the management challenges present in a changing shop, batch is the most difficult. Jobs can have any name, reside in any group, log on in any account -- not even the same one in which the job file itself resides in -- launch any number of other jobs in any number of accounts, have any number of dependencies... As you can see, the potential abuses are endless. I don't care what you have to do, but beg, borrow or steal the money and purchase a good commercial batch controller. In our shop we have had great success with MAESTRO by Computing Capabilities, but I'm sure any number of other products will suffice.
The features you will want are:

- Audit trail of all jobs run
- Automatic startup/shutdown
- Normal system security observed, with some extensions over the product's internal functions
- Paramter substitution
- Definable dependencies
- Restart/recovery
- Ad hoc reporting of job run statistics

The reason batch control cannot be overemphasized lies in the nature of batch processing. Batch is where the heavy-duty processing occurs; batch is where the big updating takes place; batch is where the morning reports originate. Unsuccessful completion of last night's batch processing can ruin your whole day! When batch processing has been stabilized, at the very least you can get a good night's sleep, which will give you the strength to cope with all the challenges of on-line processing.

-Vendor Relations-

You may not make a lot of friends with my advice here, but at least you'll avoid becoming a hot site. Your job as a manager is to be proactive. That means you avoid problems by dealing with issues before they become problems. Vendors -- including H-P -- don't always see it that way. Sure, they will schedule your P.M.'s for you; that much is there. But let's say you have discovered an excessive amount of hits on one of your drives (you do monitor your LOG files frequently, don't you?) and H-P comes out and runs COLOSSUS. Just to make it interesting, say it's Friday night. What if H-P discovers a problem that requires replacement of 10 heads and they have only three in stock. What do you do? Accept their opinion that the problem isn't serious and can wait until they can get the heads from Boise? And what if you have a System Failure #650 on that drive before Boise sends you your heads? Needless to say, you have a responsibility to your users to keep the System up and running during business hours (you can always fudge off hours). You pay good money for support -- insist on good support. Until H-P -- or any Vendor, for that matter -- will guarantee no downtime until the maintenance can be completed, you have every right to expect an adequate part supply for rectification of the problem on the spot. Believe me, once you have made it clear how you feel about support, the Vendor will take notice. And you and your users will have one less thing to worry about.

-Potential Problems-

O.K., so you're now communicating like crazy with everybody, you've archived 5,000 dead files, the remaining files are as organized as a Japanese garden, your staff is now performing only necessary tasks and in clearly defined roles, your nightly batch processing is completing successfully for the first time in years and your Vendors love and respect you. So now what's the problem? You should be aware that every solution carries some potentially unpalatable side effects.

Let's examine a few.

For one thing, you may lose some of your staff. Change is difficult for everybody, for some it is impossible. Resist slobbering over the person in every shop who supposedly knows everything and keeps it all in his head. Give yourself and the remaining staff enough credit for being able to solve problems on your own. It will be difficult, of course, but you will survive. Never underestimate the untapped enthusiasm -- and the problems this brings -- of users who long ago gave up hope of ever getting "anything" from MIS. They are now "born again" users and their expectation is that you are miracle workers -- especially if you were smart enough to satisfy some easy requests in order to establish credibility. Anticipate these expectations and, in the process of maintaining communications, make sure you are not put into a situation of committing to services you cannot possibly perform. Keep turnaround times realistic. Nobody wins when you can't deliver on impossible promises. And what about your poor old HP3000? I now have more users than ever logging on -- many for the first time. They are so excited about our concern in them that the machine is just dying from kindness. Your initial capacity planning will inevitably lag a little here, and hence some users will be -more- frustrated than before all this great stuff started occurring. You will find a way to juggle all the new demands by tuning the system and breaking jobs, etc. That is what being the manager is all about.

-Summary-

By avoiding the pitfalls of your predecessors and relying on good management practices and your good experience, change can be successfully managed with a minimum of disruption. You and your users will both learn something in the process of communication, which forms the foundation of any successful transition.

3085. Fourth Generation Languages: Use and Abuse

Mark Wallace
Robinson, Wallace & Co.
11693 San Vicente Blvd.
Los Angeles, California 90049

The topic for this paper is "4th Generation Languages: Use and Abuse", and I will be talking about the benefits that 4th generation languages have brought to the HP3000 user community. Then I'll be discussing some problems that people have had due to their use of fourth generation languages. I will suggest a solution to those problems. That solution is a formal development strategy into which 4GL's fit at a particular place and time. Another aspect of that strategy is a technique called structured analysis and I'm going to be giving you a brief introduction to what structured analysis is all about and how one goes about doing it. One of the key features of structured analysis is that it allows you to build a model of the essential requirements which your system will have to fulfill. And I'm going to conclude by talking about what we do after we build that essential model. In other words, how we take these requirements and implement them given that we have an HP3000 with certain file storage mechanisms and software and so on.

I. Benefits of 4th GL's

So let's start by looking at some of the benefits that fourth generation languages have brought to our community. First of all: faster implementation. Assume you have a system specified and designed and you need to implement it. If that system is of the kind that's suitable for a fourth generation language, you're going to be able to build it a lot faster than you would with something like a Cobol or a Fortran.

You're going to be able to build it faster because it will take you fewer statements to get the same amount of work done. Because there are fewer statements to write there will also be fewer statements to maintain over the life span of the system. And also because there are fewer statements that you have to worry about, there will likely be fewer errors in the resulting software application that you create.

A fourth generation language typically assumes or automates some of the programming that we had to do ourselves when we were writing in something like a Cobol or a Fortran. It's very comparable to the way that languages like Cobol and Fortran automated some activities that we had to do for ourselves when we were programming back in the assembler days.

So faster implementation, reduced maintenance effort over the system's life span, and a more reliable application (fewer errors), are substantial benefits of fourth generation languages.

They also have been proposed for use, and are used, in an area called prototyping. Let's look at that area a little bit more.

First of all, what is a prototype? According to my Random House dictionary, prototype is "the original or model on which something is patterned." Now in the context of a computer system, the prototype is a model which is executable. The prototype that we're speaking about runs on a computer. And the something that is going to be patterned on it, is a final system which we are proposing to build (the application software).

The prototyping model itself could be at any one of several levels of detail. You could have simply the screens themselves, just hollowed shells if you will. You could have the screens with literals and flow built in. By literals I mean, for example, for a name you might see "John Smith" just to show you where the name would go. By flow I mean we can transfer control from a master screen to a detail screen and so on. You might be able to input data and have it edited, or the screen might be full function allowing all kinds of input as well as retrieval and updating. These levels of detail on a prototyping effort were cited by a gentleman named Bernie Boar in a ComputerWorld article a couple of months ago. He's the author of a book on application prototyping.

Now, where did fourth generation languages come into this. Well, they are what we use to build the prototype. In fact, before we had fourth generation languages, it was very expensive to build a prototype, because in effect that involved building an entire copy of the system itself out of something like Cobol. That would involve just about as much effort as would be required to build the entire thing.

The advantage of the "fast build" is that we get something in front of the user. We get feedback from the user much sooner than we would if we had to wait through a longer development cycle. So we get that cross check early on as to whether we're on target or not. However, misuse of prototyping and misuse of fourth generation languages for prototyping can cause some severe problems in the application development life cycle. These problems are what I want to talk about next.

II. Problems with fourth generation languages.

One of the major ones is that they tempt us into implementing the application too soon: before we really understand what's required well enough to go ahead and implement. Other problems of misuse come when we're so enamored of a fourth generation language that we use it to build an application when there's an off-the-shelf package that would solve our requirements. Another problem comes when we use a fourth generation language when it's really not acceptable for a particular application and we really need to retreat to something like a Cobol or more likely a Fortran for the control that it would give us over a particular kind of processing. If you're doing scientific number crunching, for example, it is unlikely that any fourth generation language (that I've seen so far) is going to be of much help to you. There are borderline cases where you really might have a cleaner build if you use something like a Cobol or a Fortran.

Now the first of these problems, implementing too soon, is one that I want to take more of a look at. What do I mean by too soon? First of all let's take a look at a typical system life cycle (see Figure 1). Starting with the analysis and design stages, moving on to coding, then to a test and debug phase, and finally to the rest of the life history of the system: maintain and modify modes. This kind of chart has been reproduced in several books. The one that is probably the grandfather of them all is Barry Boehm's Software Engineering Economics.

The size of the slices of the pie is meant to suggest the percent of time that's spent in each of those stages. Now, in any given shop, the figures may vary several percentage points one way or the other for any specific phase but overall they're fairly reasonable for the industry. Of course if your development life cycle starts with coding you've already got a problem. That's the extreme case of implementing too soon and what I'm going to suggest is that there are a lot of problems with taking that approach.

Now, let me ask you a question? Given that you've got a fourth generation language, where in this life cycle does it begin to help you? Someone suggests not until we get nearer to the maintenance. Where does it begin to help you? Does anyone think it might help you a little bit before that? Okay, it depends a little bit on how you define the design phase or what activities you assign to the design phase, but somewhere around late in design to coding is where the 4th GL starts to play a role. In the analysis and in what I would consider preliminary design, it really does not provide much assistance at all. But, people who attempt to use it as a substitute for those phases can get into a lot of hot water.

Now, let me say that implementing too soon is not something that's new to fourth generation languages. It is something that has always been a temptation. In fact, we have something called the WISCA syndrome that was first identified by Jerry Weinberg, the author of Psychology of Computer Programming. Anybody know what WISCA stands for? It stands for "Why Isn't Sam Coding Anyway?" It's a typical manager's response when he or she walks into your room, now you and I are the humble programmers here, and sees that we're doing anything other than the writing of code. Because for a lot of managers, anything other than the writing of code is just a waste of time. They offer lip service to analysis and design. "Oh, yeah, we know you've got to think about it a little bit," but when push comes to shove what they really want to see is: let's grind the code out.

A lot of the time that comes from an approach which is referred to as backward "estimating". In quotes because I don't really want to dignify it with that kind of a name. The way it works is the manager comes in and says "I've promised the user that we're going to have this by December 1 and I know from my years of experience in this industry, sonny, that the only way you're going to do it is if you start coding on it tomorrow, so by gosh, that's when you better start coding." We take an arbitrary deadline and work it backward to when coding has to begin.

Well, of course if you don't know what the problem is you're supposed to be solving that can cause you some severe difficulty, as you might imagine. Now, fourth generation languages have come into play as items that cause a temptation to do this more so even than the third generation. If you're using a fourth generation language, you should be able to wait longer before you start coding, right? Because you know when you get to coding it's going to go faster.

However, if your manager is not familiar with that kind of speed up, he or she may resist that bargaining or request on your part and say, "Well, you better start it June 1st anyway because I've been around for a lot of these projects, sonny, and I know that you can run into problems and I want to see that thing done by December 1st." So the pressure to get into programming or prototyping can become great and, not only that, the pressure to then let the prototype become the final system, which it was never intended to be, can be almost irresistible. But there are some real problems with that which have been reported in the industry.

Here's a series of excerpts from an article in ComputerWorld referring to a prototyping facility at the Bankers Trust Co. in New York. They use a fourth generation language called Focus which is actually probably a step more comprehensive than most anything we have to work with on the HP3000 It will interrogate you and construct the data base for you and allow you to build an entire application from scratch and is supposed to be fairly user friendly and so on. It's a main frame capability package, very expensive, very capable.

But what happened at Bankers Trust? Programmers referred to as "Focus acrobats" tended to attack requests without proper analysis. And these are your hackers who graduated from high school and now they have a job with your company. They're bit pushers who really get their enjoyment from twisting and turning these fourth generation languages to do things that would otherwise be difficult to do. People like that have a place in the company but we have to keep that place well defined. The problem with letting them make that initial step too soon is that it "led to applications that were poorly designed and documented and had data integrity problems."

Now, a little while ago we saw Bernie Boar's view of prototyping. Here's this gentleman's view. "A prototype is a full working representation of a final system, not something you throw together in an afternoon. You should be able to put production data into it and run in a production mode." He has a much more fully formed view of what he means by a prototype.

He goes on to say, "poorly designed prototypes were being used as a base to create poorly designed systems." Furthermore, and to me this is critical, "users were losing interest in confirming systems specifications since they always assumed the prototype could be changed."

We get these vendor claims that, or people read into vendor claims the idea that, you can take your 4GL, toss something out and if they don't like it you just rework it, until it is what they like. But that reworking can take a long time. What these people ran into was: it was taking longer to do it that way after half a dozen false starts than it would have if they had started out and specified this thing from scratch.

Let me give you another example. A significant size system was implemented recently by Hughes Aircraft Radar Systems Group. Their Kurt Lysy reported at the Structured Development Forum in Newport Beach earlier this year that prototyping, as a result of the experience at Hughes, at least in this group, was not a substitute for analysis and design. We simply cannot identify the functional requirements of the system as well as a careful analysis and design.

It places too much emphasis on the details of form and screen layout. You get the prototype up in front of the user and you get the early user feedback, that's true. But what kind of feedback is it. I'd like to see this field moved from column 15 to column 17. Make this one blinking instead of underlined. I want this column to have a column heading that's stacked instead of two words next to one another. That, I would like to submit, is not the kind of feedback you need at this stage. Those are coding details that can be dealt with a lot later in the application development cycle. They take your attention away from the policy requirement, the essential requirement that the system has to fulfill.

Next problem. Prototyping is dependent on a pre-existing stored data model. All of these fourth generation languages and prototyping tools require that the data base or file structure be there already. If you don't know what it ought to be, you're not going to find out very conveniently by tossing up prototype screens. That's going to be a very painful and slow learning process. Like when the feedback requires you redo your data base or file design, that's going to be a slow and painful process.

Last problem: the temptation to take the prototype and "enhance" it into the final system. I say enhance, more often it's warping, twisting and turning. It happens if either you're late or politics come in. User says, "Hey, you know, all the screens are there, just stick a few more things in and we'll be done." Not very often the case.

What is the benefit of prototyping as discovered or confirmed by Hughes Aircraft? It does provide valuable assistance in defining the man-machine interface. Your screen layout, your form layout, what the system looks like to the user. It's great for that. You can change things around very quickly once you've got a solid base to build on.

So, at the risk of inflicting another acronym on an already over-burdened industry, I would submit that we have to deal now with not only WISCA - "Why Isn't Sam Coding Anyway?", but also with WISPA - "Why Isn't Sally Prototyping Anyway?" From the manager who has just been visited by your friendly vendor sales rep and been convinced this package is the answer to all of your needs, dreams and prayers. "Let's get with it." Well, that could be premature.

Given that we have these problems with a premature use of fourth generation languages, what would I care to suggest as a solution? Well, as I mentioned earlier I'm going to propose a formal development strategy for building applications. I will first attempt to provide some justification for that, then I will give you an overview of a specific strategy that I would suggest, although it's by no means the only one. A key piece of that strategy is a formal requirements definition stage, also known as systems analysis. I'll mention some problems with the traditional approaches to systems analysis, and what I would suggest is an answer to those problems, which is one particular technique known as structure analysis.

III. Solution: A Formal Development Strategy

A. Justification

So let's look at the solution and why I think we need, and I'm not the only one, a formal strategy. Bankers Trust from the same ComputerWorld article: "Prototyping should be used within a systems development methodology. At times it may not be appropriate at all." Many of you may have been privileged to hear Ken Orr speak at a previous SCRUG meeting where he refers to the 'cut and run' approach to systems development. Again the same kind of philosophy that we start prototyping before a formal requirements definition.

There's a very interesting introduction in the SCRUG newsletter for the 1985 conference, written by Maureen Nott from Intel, who was one of the speakers on capacity management, planning and tuning and so on. She says, "With the increased power of the HP3000 has come an increase in workloads. (Here's the key.) Larger and more complex application drive us to look beyond the familiar day to day systems tuning to search for more concrete methods of resource management." What, I would like to suggest is that the same factor, larger and more complex applications, is also driving us toward more concrete methods of software development management.

As we get into, with systems like the 68, a ballpark of applications that the HP world hasn't seen before, perhaps these statistics from Capers Jones may be a bit sobering. From ComputerWorld last November his prediction is, and based on statistics that he's collected, up to 25% of large software systems (those with over 64,000 lines of source code), currently under development will be canceled before they are completed, and up to 60% will experience significant cost and schedule overrun.

64,000 lines of source code is pretty big for an HP3000 application. Generally, when you see something that size, you would be talking about some sort of general purpose package but people are creeping up in that direction. The management is not buying those 68s just to run the weekly football pool. They want to see a lot of work out of them. So that's some justification for a formal strategy.

B. Strategy Overview

Here is one possible development strategy that you could take (see Figure 2). It's not the only one. We start off with an activity called the blitz that I'm going to cover in more detail later. Basically, that is a real fast sweep over the requirements in order to get a feel for the overall scope of the project. That results in a preliminary model which in turn is fed into two separate activities: an information modeling phase and a formal specification phase.

The information modeling phase yields what I call an ERA or Entity Relationship Attribute model. Basically it's a model of the data that you will need to store in order to satisfy the output requirements of the application. The specification phase produces what I call a structured specification. If both of those are input to design, that produces a blueprint for the building of the system and in turn, that's used to implement the system, which generates a running version.

In the meantime, the spec has been used to generate test data independently of the design and implementation activity. The test data and the running system are used by a quality assurance function to verify that in fact everything is as it should be. And, the results of that are used by an installation activity to decide whether or not to deliver the resulting system. This is not the only approach. Mixing and matching is possible for the needs of a particular organization and a particular application, but it's certainly one that has well served users who look for a formal development strategy.

C. Requirements Definition

I want to focus on the phase that I call "specify". That's where we're going to define the requirements for the system. Requirements definition is also known as systems analysis. The purpose is to establish what the system must do. How to do it is something we're going to defer until design. Again going back to Bankers Trust and their experience. "What is required is a functional specification that is approved by the user. In fact a project cannot proceed to the next phase without it."

Another article in ComputerWorld last May by Maureen Lampirello, "The lack of complete and correct specifications is the problem. All too often project costs are estimated and completion dates are targeted before the goals and boundaries of a project are fully discussed. In the worst cases, work actually begins on program design or coding when the project is little more than a sketchy statement of the user's requirement." The danger with this is that you may fall victim to Gordon's Law, which states that "if a thing is not worth doing at all, it's not worth doing well." So let's not spend a lot of time and energy designing and implementing something about which nobody cares. Let's identify the requirements first.

Another slice over the same idea is that "pay me now or pay me later" commercial. If you want to skimp on analysis and design and go straight into coding you're going to pay the price. But the difference is that the cost to make a change in each development phase goes up by approximately a factor of 10. In other words, if you deliver a specification to the user and they find something in it that is not what they wanted, let us say hypothetically that the cost to fix it in the specification is $100. Given that that was so, industry statistics indicate that if you have completed the design before the discrepancy is noticed it's going to cost you $1,000 to make the change, and if you have coded and implemented the system, it's going to cost you $10,000 to make the change. So the idea is, if you pay me later, the tab is going to be a lot higher.

D. Problems with Systems Analysis

Given that we want to undertake a formal systems analysis, what are some of the problems that we can run into? First of all there's a personnel issue. The skill set that's required from an analyst is very different from that required from a programmer. Some companies are making very good strides at getting users involved in doing systems analysis. DuPont on the East Coast is one. But, for the most part, people who do analysis, like myself, came up from the ranks, from a programming background.

As we all know, a programmer has to be able to communicate with the computer. There are a lot of us around who, at least at one point in our lives, were not particularly good at communicating with people. Maybe that's why we even chose programming as a profession. An analyst, on the other hand, is a very different story. There you have to communicate with people, both the users on the one hand and the technicians on the other, who are going to build the system. So promoting someone with a programing background without giving them adequate training in how to do systems analysis can be a source of problems.

A company, which is an HP3000 user with which I am familiar, had a systems development project going on about a year, year and a half ago, and they committed to systems analysis. They said, "we're going to study the requirements before we build anything," but they had an analyst who came up from a programming background, had no training in how to do analysis and they ran into some major problems. The analyst went around talking to users for six or eight months or more and never wrote anything down in any formal way, just scraps and notes and stuff, and never organized their requirements in any systematic fashion.

So, when it came time to do some designing and building of the system, they were very poorly positioned to go ahead with that. The analyst and the designer on that project are no longer with that company and the project was shelved the last I heard. So, your analysts have to be trained, especially if, like me, you come from a programming background.

But a lot of the training that's available is teaching analysts to do things in ways that have since been demonstrated to be outmoded. They teach you to produce specifications that are redundant: the same pieces of information are defined over and over again. They are wordy, long narrative descriptions of what the system's supposed to be doing. They are too physical. Concentrate on things like report layouts again, or screen design, or the specific technology that's going to be used to build the system, specific applications of computers.

And as Tom DeMarco, one of the leading experts on analysis, has commented, they are tedious to read and unbearable to write. What happens is you get a document that's about as thick as your conference notes, and it gets dropped on the user's desk and the comment is: Read this and sign it in blood or we're not going to go ahead and build your system. Well, the user might sign off on it, but I think a lot of you will agree that it is hardly what you would call informed consent.

E. An answer: Structured Analysis

What's the answer? Well, I've kind of given it away by telling you that in the next section we're going to be talking about structured analysis. Tom DeMarco, who was one of the pioneers of structured analysis, when he set forth the goals for a specification that his approach would produce, wanted it to be maintainable. He wanted it to be partitioned in a top down fashion, so that you could look at one piece of paper and get an overview of the entire system, and then if you wanted to pursue any subsystem on other pieces of paper to get more detail.

Another goal is to use graphics - a picture is worth a thousand words. And you'll see some examples of that very shortly. In fact, you already have seen some examples. And he stressed the need to differentiate between the logical and the physical aspects of the system. The logical aspects are the requirements that are essential to the functioning of a system regardless of what level of technology you use. For example, if you have a payroll system, you're going to have to generate checks - paychecks. You could do it by hand. It could be a completely manual system - pen and paper. Or it could be totally automated with the latest state-of-the-art, on-line data entry and so on. Whether it is manual or automated or what technology is involved - that is a physical aspect of the system. We need to keep that separate from the functional requirements.

Now the approach that structured analysis takes is that it encourages us to build a model of the requirements. What do I mean by model? I searched my Webster and I got to the eleventh definition of the word model and I came up with one that I was happy with for this context. "A description or analogy used to help visualize something (like an atom) that cannot be directly observed." Well, I would submit to you that your new system cannot be directly observed yet because it doesn't exist.

So what are we going to do? In requirements definition we are going to build a model of that new system. A model will omit some aspects of the system, otherwise it would be the full system itself. What are we going to omit? We are going to omit things like screen format, report layout, physical storage detail, and the sequential flow of control. Those will get incorporated at a later point, after we have defined the essential requirement. Now notice that those are the sorts of things that prototyping focuses on right from the start. And that again, is another slice at why prototyping is not a substitute for requirements definition.

IV. How to do Structured Analysis

What I'd like to do now is give you a very brief introduction to what structured analysis is all about. What kind of models are we talking about building? Now, this sort of thing is taught over 10 to 15 days of training by various organizations so in the 10 to 15 minutes that it takes you to read this obviously I am not going to be able to turn you into fully capable structured analysts. But I wanted to give you some kind of flavor for the tools that are involved and the techniques that are applicable for putting those tools to use. Tools are the data flow diagram, the analysis data dictionary, mini-specifications and then a concept known as leveling of data flow diagrams for systems bigger than a certain size.

A. Tools

We'll start with the data flow diagram. You've seen a couple of these already in this paper. The data flow diagram has four different kinds of components on it. We've got the arrows along with the arrow heads. Those are data flows. They represent data or information flowing from one part of the system to another. We've got the circles or bubbles. Those represent processes or activities. Those will typically receive some data and transform it into a result. We have the double parallel lines which are used for data stores. Data store is simply a technically neutral name for a file or data set. It could be Master, Detail, KSAM, MPE, whatever. And then we have the boxes, which are either sources of data flows or sinks or receivers of data flows, or a given box could be both as in a person who submits an inquiry and gets back a response.

Now this data flow model of the system has some interesting properties. First of all it is a network model. And by that I mean that all of the processes can be thought of as active at the same time. You can think of it like an automobile assembly line. The automobiles start out at the end of a line, let's say start out with a frame, and then maybe we attach the wheels to the frame and it moves down and then we drop the body on, and then it moves down and then the engine gets dropped in it, moves down to the next station and the doors get put on, and so on. Now, as the car leaves one station and goes to the next another car comes to the prior station, so each of the stations is active simultaneously.

But of course any one car is going through it sequentially. Now the benefit of that is that a car pops off the other end every 60 seconds or so instead of every half an hour, which is how long it takes that car to go from beginning to end. So all of the processes are active simultaneously but any one transaction coming into the system may only go through one at a time.

Also, it's a steady state model. We don't deal with initialization or termination, we assume the system is up and running and will continue to do so. That's not to say that the system will not have to be initialized, it's to say that that is not something we're going to think about now. We're going to defer that until design time. Similarly we're going to ignore control flow until design time. Interrupts are pure stimuli as opposed to the data flows that we do care about.

Now this data flow diagram follows the principle that we do not want to overload any one component of our structured model. So if a data flow, for example, was an order for goods we would just call that an order here. We would not list all of the data elements that make up that order, like what goods, how many, who the customer is, where to ship them and so on. We have a summary name and at another place in the model we will provide a more detailed look at that particular component. So, the advantage is, we can see the forest here for the trees.

Here is an example of a data flow diagram (see Figure 4) for a very simple order entry subsystem that I just pulled out of a class that my partner and I teach on one particular fourth generation language. You've got a customer; customer can make an order. We have an activity that accepts the order. In doing that, we check the customer file to be sure that the customer ID coming in on the order is a valid one. We're going to check the parts master to make sure that all the parts being ordered in fact exist. We'll check the sales master to validate the code of the sales representative who booked it, and we will create an order header record, and multiple line item records, one for each part that was ordered. We will generate a packing slip that we send to the warehouse, where presumably they will pick the goods off of the shelf.

When a customer makes a payment we will accept that, we'll update their current balance in the customer file, and we'll create a record that we will add to the payment history file reflecting the fact of the payment. This is a high level look at what the system does.

Here's another data flow diagram (see Figure 5). This one is from another one of our classes. Notice here that the network nature is brought out because we've got, for example, off of a compiled dictionary the ability to run any one of about six separate tasks at the same time: Quick, QTP, Quiz, and three utilities associated with the dictionary. All of those can be active simultaneously, so again the model gives you a picture of that parallelism which is very difficult to convey if you are writing a sequential text description of what the requirement is. That's why this "picture is worth a thousand words" thing is a real benefit.

The data flow diagram can be checked for syntax. For example, if there is a bubble into which data flows but nothing ever comes out, you don't have to look very far to suspect that something is wrong. You can also check it semantically. If you've got a bubble into which flow sugar, eggs, and water and out comes orange marmalade, a semantic understanding of cooking will suggest that perhaps something is missing from the input. That, you couldn't determine just from observation, you have to understand the nature of the activity.

Now, given that the diagram is a high level model, where do we turn to get more details. The first place is what I call the Analysis Data Dictionary. Now, structured analysts refer to this simply as the data dictionary but I want to differentiate here because there's a difference between this and the data dictionary that you may be more familiar with. In the Analysis Data Dictionary we have an entry for each data flow, which breaks it down to the data elements that are carried on that flow.

For example, an incoming order might be your customer I.D., the date the order was placed, and then for each part the part number and how many they wanted and so on. Also, for each data store, there will be a definition of the elements that make up that store. So we're indicating the composition of each of these things. We are ignoring physical storage details. We don't care whether it's packed, or binary, or signed or unsigned, or anything like that. We don't care about the formatting, whether we're going to float in a dollar sign, stick in a comma for every three digits; all of those again are design and implementation considerations.

So the reason why people who use what I call run time data dictionaries, like Dictionary 3000 for example, have some problems with structured analysis is they tend to fill in all the blanks. Dictionary 3000 or the Powerhouse dictionary or, I'm sure, many of the others can store a lot of information about each data element, that is completely irrelevant to the requirement. It will be a factor when we get into design.

Here are some sample data dictionary definitions. They tie back to the earlier diagram (Figure 4).

```
ORDER = ORDER-HEADER + {LINE-ITEMS}

ORDER-HEADER = ORDER-NUMBER + CUSTOMER-ID + DATE-PLACED +
               (SALES-REP-CODE)

LINE-ITEMS = ORDER-NUMBER + PART-NUMBER + QTY-REQUESTED

PAYMENT-TYPE = [CASH | CHECK | CARD]
```

Order, can be defined as the order header plus some number of line items. The braces indicate that we have the possibility of repeated occurrences. Then we break down order header into order number, customer ID, date placed, and then the parentheses indicate optionally a sales rep code. Maybe it's a house account and there is no sales rep. Line items are broken down into order number, part number and quantity requested. Payment type might be, and here the square brackets indicate choose one of, either cash or check or card, meaning credit card. So we have some symbols that we use to compactly represent these dictionary definitions. And those explain the data flows and the data store.

The bubbles are explained by mini specifications. We call them mini specifications because we've used the bubbles to break down the system into miniature systems. We don't have to specify the whole lump at one time. Just a piece at a time. These mini specs are rigorous descriptions of the activities carried out by one bubble. We want to focus on the policy that's being applied by that bubble, not on the programming techniques. The mini spec is not the coded implementation. Strategies like decision tables or decision trees can be very useful for mini specs.

If all else fails, you can fall back on something we call structured or disciplined English and here is the mini spec for the accept order bubble:

If Customer ID does not exist, reject the order Otherwise, create an Order Header record, storing the input order header data. for each input Line Item, if Part-Number does not exist, reject the line item otherwise, create a line item record, storing the input line item data.

This one is fairly simple because you don't have a lot of calculations being applied like calculating a discount or deductions from a paycheck and so on, but if that were the case, this is where those would be specified. So again the bubble on the diagram is a high level view of the mini-system.

Now, we want the system to be defined or modeled on one page. How many of these bubbles could we fit on a single page? Somewhere around a half a dozen to maybe eight or nine, we're going to have as much on there as we really want to deal with in a single picture. Suppose our target system has more than 8 or 9 essential activities? Well, what we do is, we zoom in on a single higher level bubble and create for it a lower level diagram explaining what goes on inside it, and we can repeat that. On the lower diagram we can zoom into one of those bubbles and break it down to another diagram. At the top we have a context diagram that shows the entire system.

Now let me give you an example of that (see Figure 6). Again drawn from one of our training classes. Here we have an application system that happens to be written in the Quick package. We show the programmer and the operator interacting with the system: referencing data stores, the dictionary and the user data, with the data flows as shown. Now this is a context diagram. The entire system is in this one bubble.

Well, suppose we want more detail. Let's zoom inside that bubble and see what's going on (see Figure 7). Here we see inside that bubble there are three smaller ones. QDESIGN, which is, if you will, the compiler that translates your Quick specifications into an executable screen. We have the Quick driver that executes the screen at run time. We have a facility called SetQKGO which simply provides some parameters that Quick will use at run time. The same incoming flows from the operator and the programmer are shown on this lower level diagram. We simply break the handling of those incoming flows down to a greater level of detail.

If you want to, you can look inside one of these bubbles, so let's look inside the Quick bubble (see Figure 8) and we find a lot of separate procedures that are used by Quick at run time to implement its processing, and I won't go into the details of those, and you could take it down even additional levels still (see Figure 9).

Now, just to review the tools, what you will end up with is a levelled set of data flow diagrams indicating the processes that the system carries out, the data flows into and out of those processes, the data stores in which we keep information within the system, and any sources and sinks in the outside world, by that I mean outside of the system, with which we have to communicate. And then to review, the processes are explained in detail by either a child diagram where we zoom in on a bubble or a mini specification where we write the detailed policy that the bubble carries out. The data flow and data store, each will be explained by a data dictionary entry. Now that's the kind of model that we want to build.

B. Techniques

What sort of strategy or techniques can we use to build it? Again, just one possible strategy, and here I'm using a data flow diagram to model the process that a structured analyst carries out in defining a system (see Figure 10). We start with the user, we get information about the existing system, we model the current physical system. We then derive the essence, the essential requirements; in other words, we strip off any signs of the technology or particular hardware that's being used. That gives us a current essential model.

We get the new requirements from the user. We derive the essence of the new requirements. We merge those into the current essential model, to come up with a new essential model. This is the logical requirement for the new system. We take the new physical requirements and the new essential model, and we carry out an activity called new physical modeling which actually becomes almost part of design. It's getting harder and harder to draw that boundary. In fact, fourth generation languages are one facility which is making it harder to draw that boundary. And at this point we consider any new constraints in the nature of the hardware and software that we'll be running on.

Now why do we care about the essence of the system? Why do we care about the essential requirement? Let's consider what we mean by the essence. In essential modeling, we're going to use some basic principles. One is technological neutrality. We don't want to presume any particular system. I just saw in an article in ComputerWorld a few months ago, IBM is shutting down its last plant that produces punch cards. 80 column cards. This is just a few months ago. They're closing this down. People still use them, right, but not enough for IBM to justify keeping the plant in operation.

A large part of the reason for that is that when people upgrade their systems, they don't take this intermediate step of identifying the essential requirements. They just convert from a 370 to a 4300, or a series 3 to a model 48 or so on. You up the technology but you don't take that opportunity to look back at what the system really needs to be doing.

In essential modeling, we will assume that the internal technology that we have to implement the system is perfect, that our processors are infinitely fast and have infinite storage, for example. And of course, if you listen to your sales reps you may sometimes get that impression, but we're not quite there yet. That's why we have to have that follow-on phase where we do the physical modeling.

The benefits are, we focus on the policy requirements, we eliminate any physical fossils like a system where all the records are 80 characters long. Has anybody seen any of those lying around? We avoid a premature focus on the new implementation. This is how we get the maximum confidence that the new system will do what the user wants it to do.

Now, in breaking down the essence of the system we're going to break down the processes using the idea of event partitioning. We're going to identify the events to which the system has to respond. An event would be either an external event like "customer submits an order," or "supplier delivers goods," or a temporal event like "time to issue the paychecks." Once a week, we know we have to put the paychecks out, and we don't need somebody to come in with an ad hoc suggestion that we do that.

We will have one high level bubble which contains the entire system response to each event that we identify. The data we'll break down using the idea of object partitioning. The object will come either from the information model or we can synthesize them from the stored data requirements in our structured specification. We will end up with one data store for each object. This is still a logical model. When we put it onto a computer there may well be multiple data stores for a given object or we may have to collapse more than one object into a single store.

I talked about starting off this modeling activity with something I call the blitz. That is nothing more than a fast build of the essential model. It can be done, it should be done, in no more than a week or two even for a large mainframe system. For an HP3000 system, you may be able to blitz in one or two days rather than one or two weeks.

What are the speed-up tactics that you use during the blitz? Well, you would stop the model one or more levels up from the bottom-level bubbles. In other words, don't go through all the details during the blitz. There'll be time for that later. You can also take short-cuts with the data dictionary entries, and the mini-specifications. For example, don't create data dictionary entries when sample forms are available. Also, don't write detailed mini-specs if you can cut and paste from existing policies and procedures manuals.

The purpose of the blitz is to get a rough feel for the size of the project. Why do you need one? Well, you can use the blitzed model to estimate the time and cost to do the complete analysis. It can be used as a guide to allocate personnel and system resources during later project phases. And, it gives early warning that you may have to either re-scope (shrink) the project, or plan to deliver it in incremental versions. But please remember that, just like the prototype is not the final system, the blitz is not the final systems analysis.

V. Life after Essential Modeling

After you have completed the New Essential Model, what next? What happens during this phase we've called "New Physical / Design?" Well, to start with, you have to identify the candidate physical processors for the new system. This includes both manual (human) processors and automated ones (computers and software). You take a census of those processors and their capabilities, and try to match them up against the essential activities from the New Essential Model. That allows you to allocate the essential activities among the physical processors. You will then, typically want to optimize the physical model, taking into account the constraints that those real-world processors operate under.

At the same time, the stored data model is being converted into an actual data base and/or file design. Again, optimization would be done in consideration of the limits of your particular data storage technology, both hardware and software. These would form the components of the New Physical Model, and, from there, the software design "blueprint."

The final phase is implementation. Be sure to pay attention to manual procedures as well as the automated ones. Many projects have been sunk by antagonistic users when a little effort up front from the analysts to anticipate their needs would have saved the day. The automated procedures will be coded in the most appropriate programming language, and screen and record layouts will be generated for review by users. This is where 4GL's can really help. You can revise these layouts very quickly to provide better human engineering and more user-friendliness.

Some of you may be thinking, after all this, "We don't need any of this stuff. We're going to buy a package." I have one question for you: "How do you know you're getting the right one?" I know, personally, of more than one HP3000 user who bought a very expensive manufacturing package (I won't say from whom), and was totally unable to implement it. They ended up switching to another package, after two years of wasted effort.

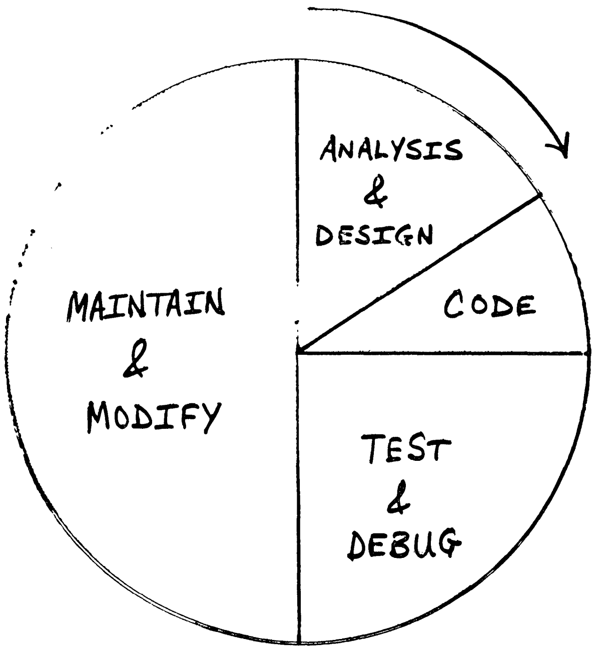
John Palmer, one of the co-authors of Essential Systems Analysis, has written a paper entitled: "Applying Structured Analysis: The Acquisition and Installation of Software Packages." His point is that you need just as much, if not more, to go through a complete requirements definition before purchasing a package. Once you have this kind of requirements definition, purchasing becomes much easier. You just release the specification to all competing vendors. "No marketing glossies, thank you. Just a bubble by bubble response as to how well your package fits our needs." And where no package is an exact fit, you have a super tool to estimate the time and cost to modify one to handle your unique needs. Can you see that you'll have a much better structured procurement cycle with this kind of approach?

Finally, end-user involvement. Again, from Hughes Radar: Can prototyping allow the end-user him or herself to build the application? The conclusion for a small system: Probably we're getting to that point. For a large system: Pretty doubtful. And not only the size but how the system integrates with the rest of your applications is a factor. If it has to communicate with a lot of existing stuff, again, turning the end-users loose to prototype is not going to be your best move. Like turning a lot of them loose with separate micros can cause a problem in data integrity and integration. However, there may be a political benefit to letting them try to build their own application. Right? You know, give them a lot more of a gut feel for what we suffer through.

Finally, to summarize what we've covered in this paper: 4th generation languages are great tools, but like any other tool, they need to be used where they are most appropriate. I feel that prototyping with them is not a substitute for a careful systems analysis. You need a methodology for building software. Structured analysis has proved to be an effective tool for requirements definition. It should be used even when you're pretty sure you're going to solve the problem with a package.. Otherwise, how do you know you're getting the right package? And, how do you know what you're going to have to do to customize it for your needs? Finally, end-users are still not able to implement complex systems by themselves.

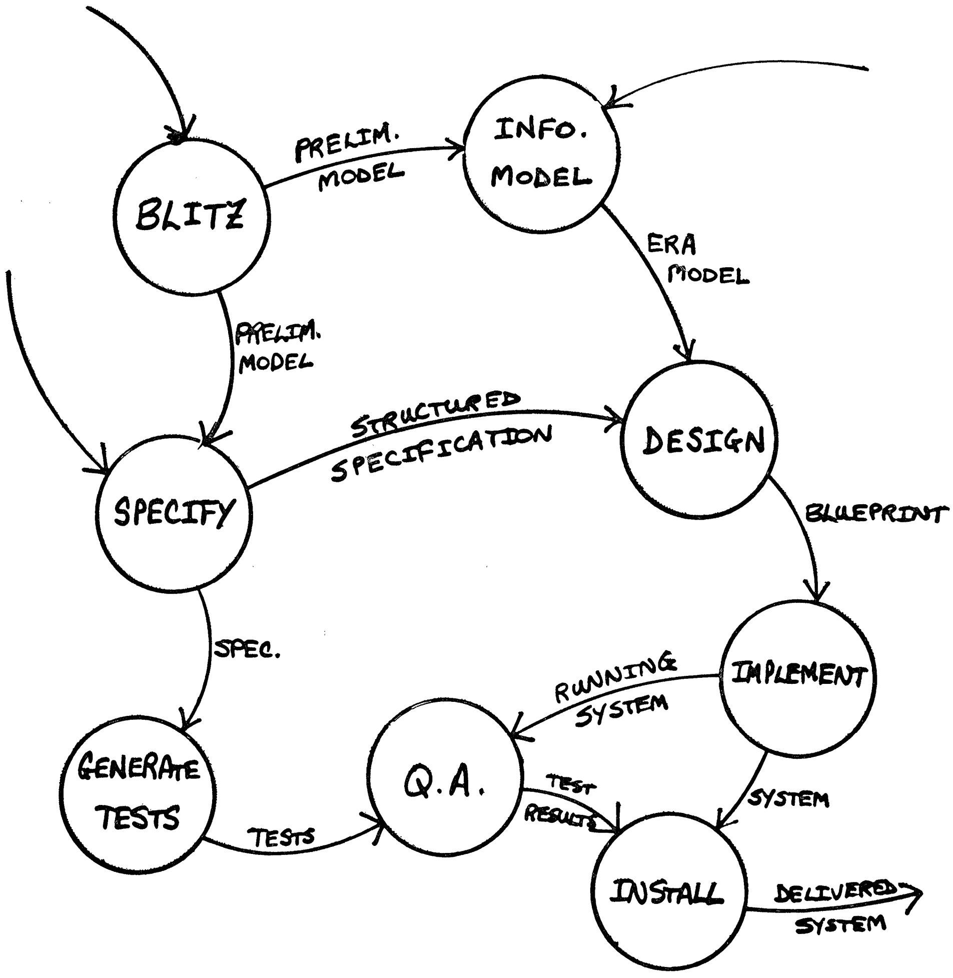
Figure 1

SYSTEM LIFE CYCLE



Barry Boehm, "Software Engineering Economics"
Prentice-Hall, 1981

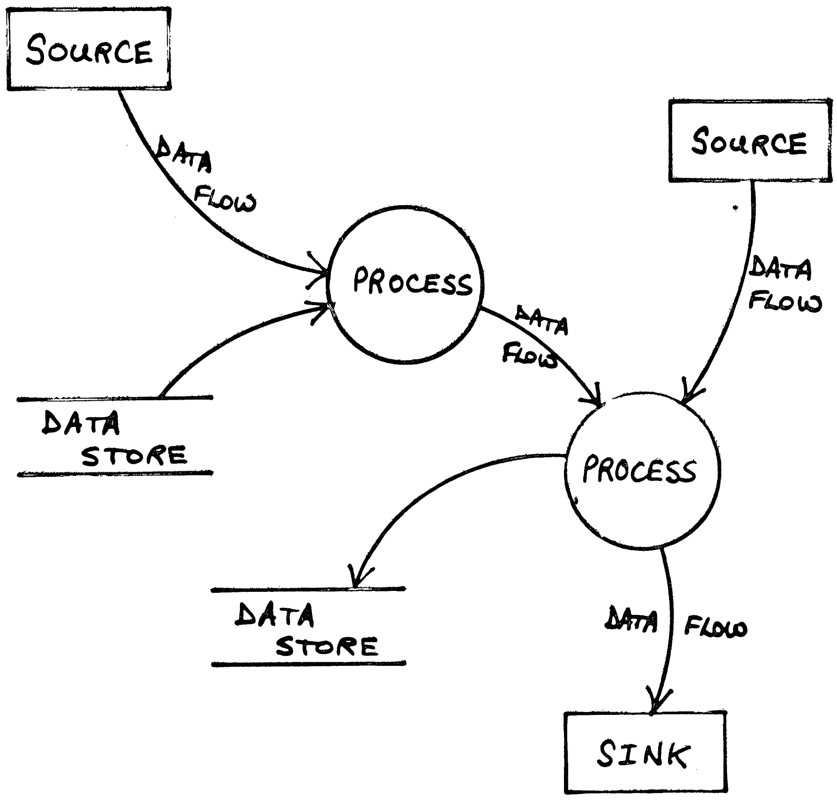
Figure 2



STRATEGY OVERVIEW

Figure 3

DATA FLOW DIAGRAM



- Network model
- Steady-state model
- Ignore control flow
- Don't overload any one component

Figure 4

Order Entry Subsystem [DFD]

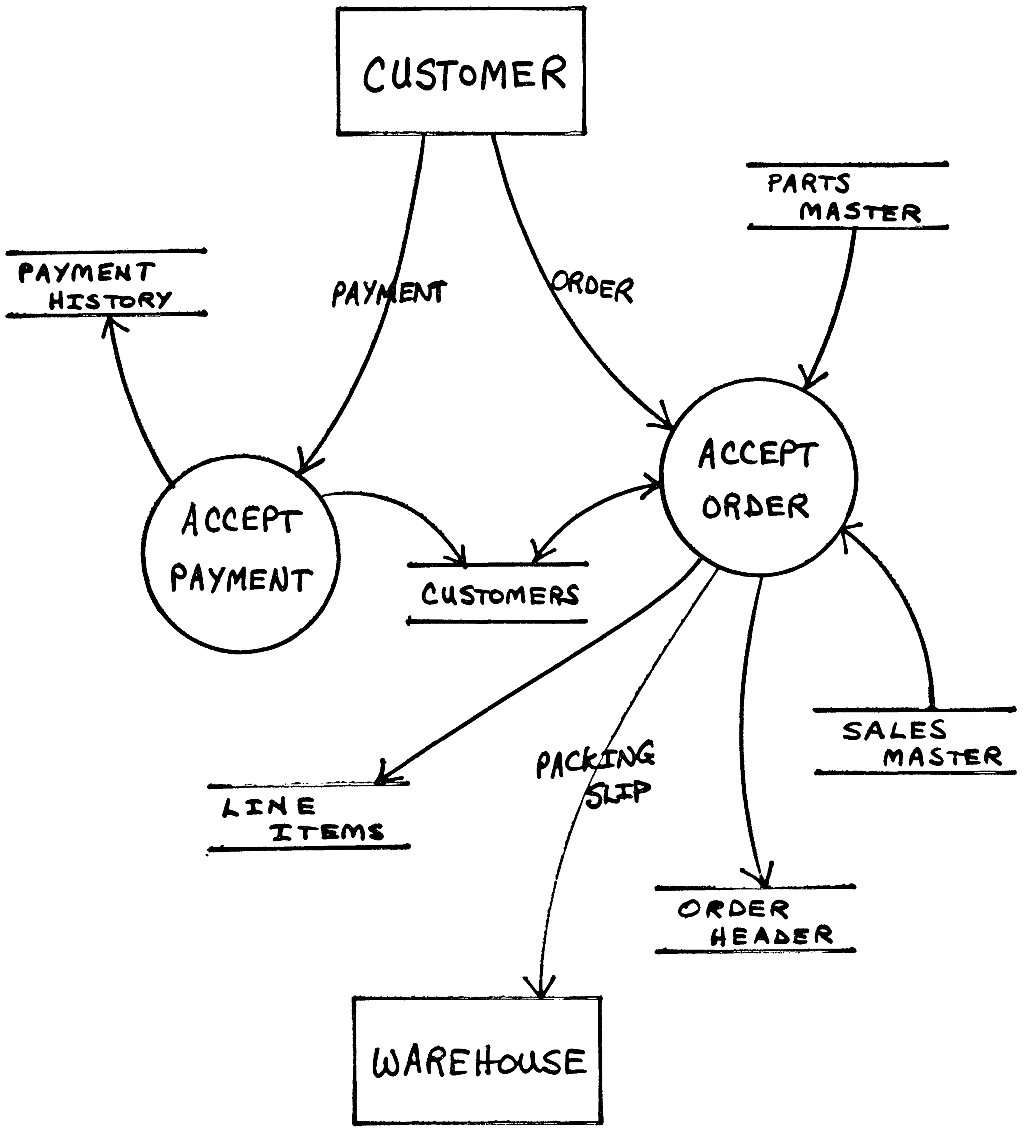


Figure 5

DICTIONARY – Data Flow Diagram [DFD]

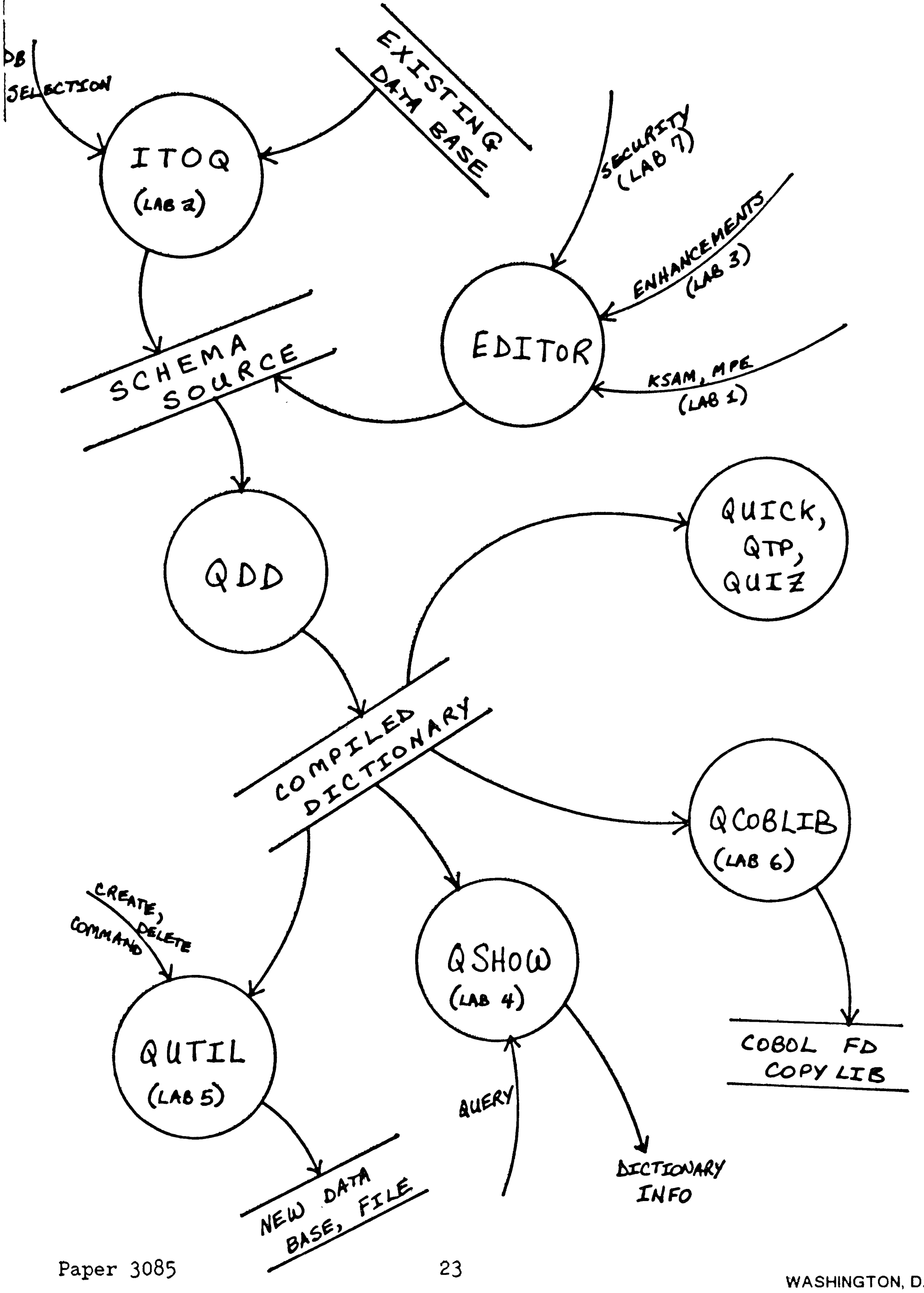
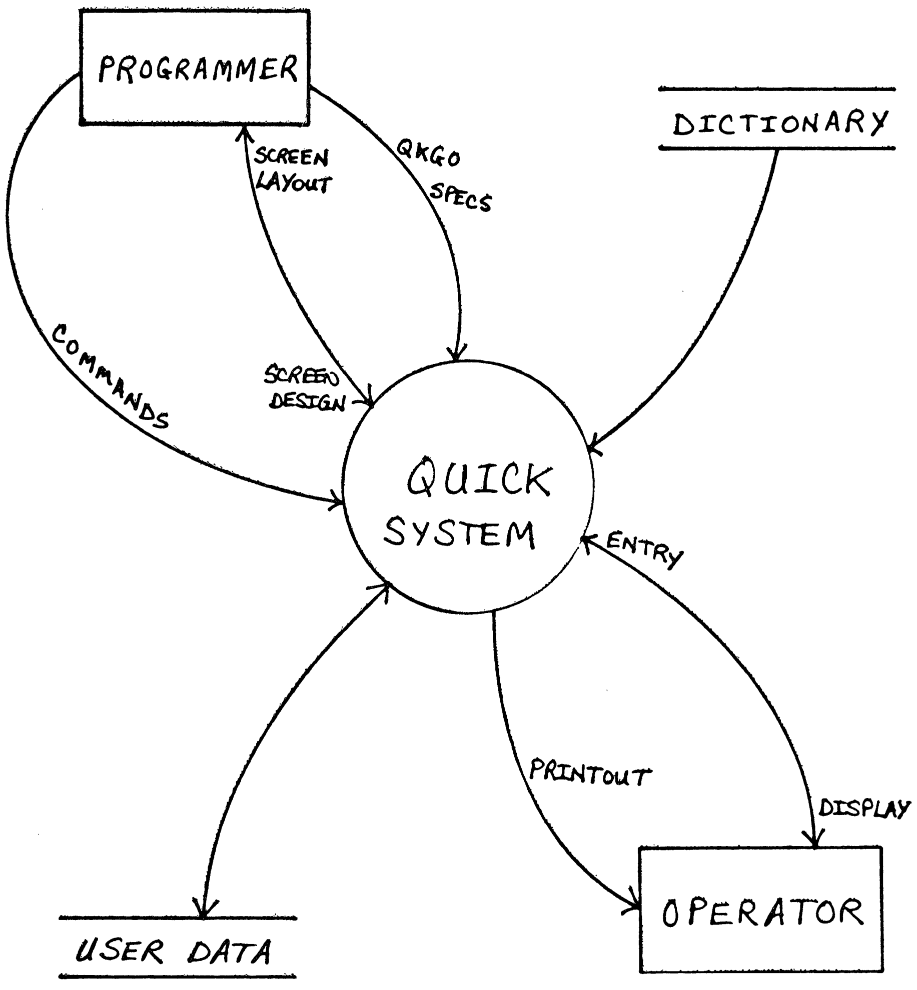


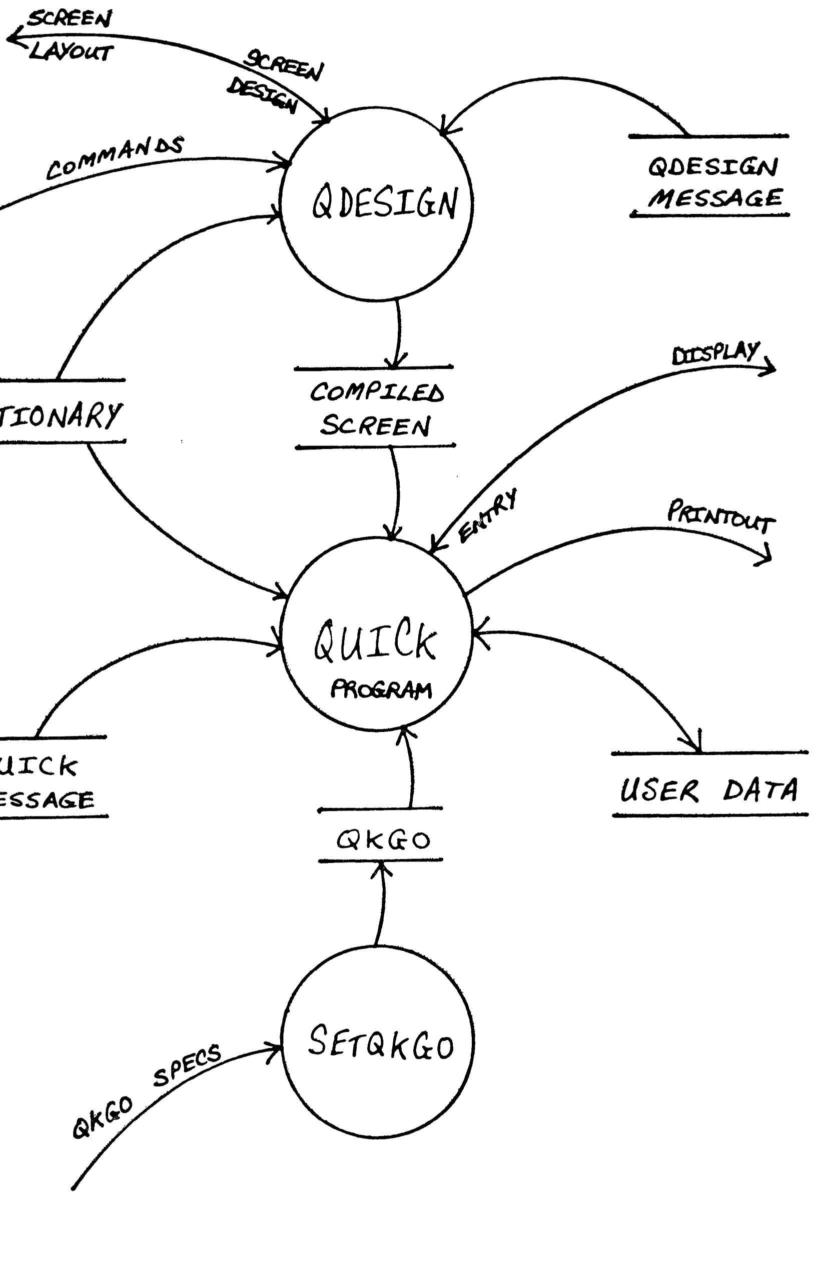
Figure 6



CONTEXT DIAGRAM

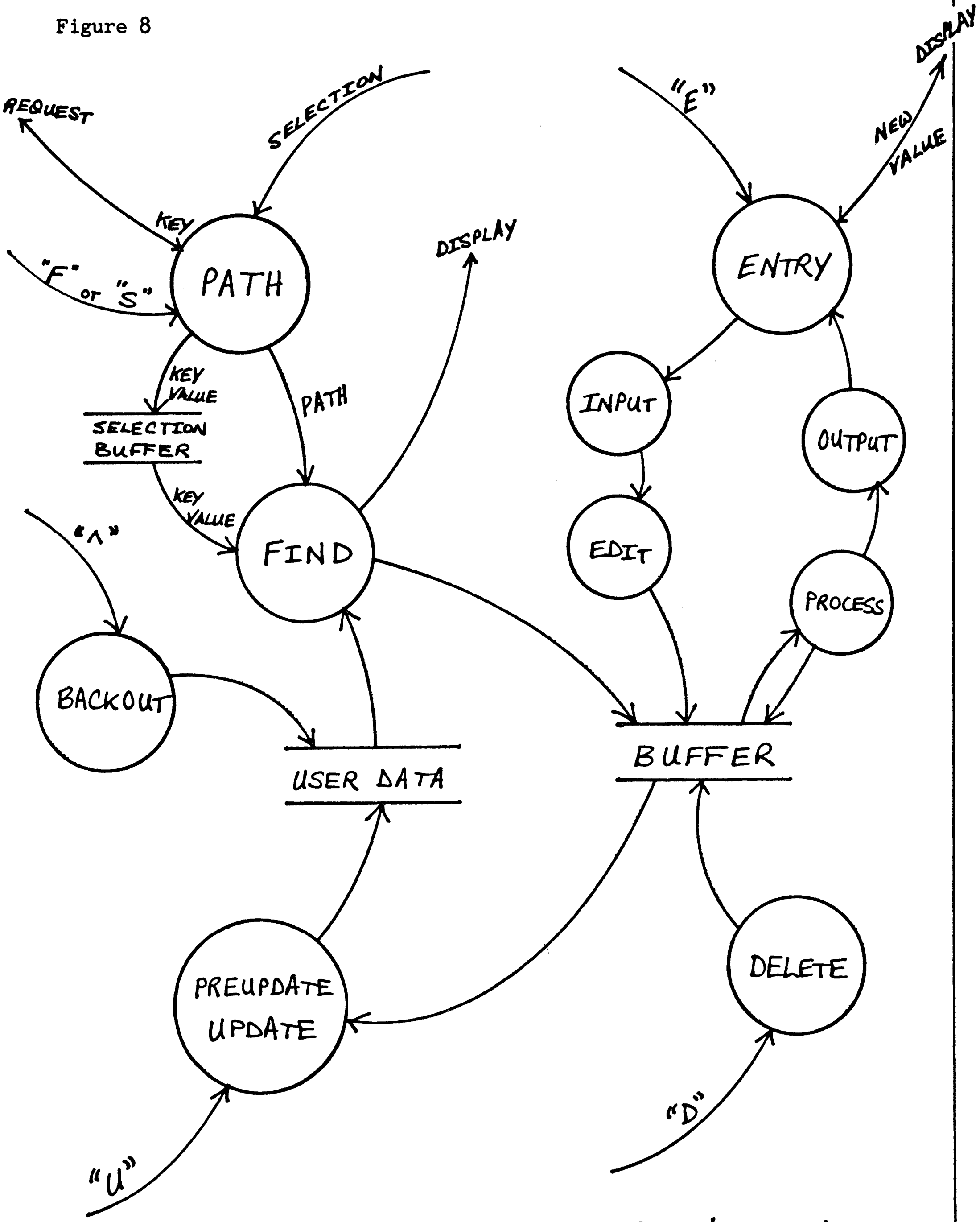
ROBINSON, WALLACE & COMPANY: Page 4

Figure 7



QUICK – Major Components

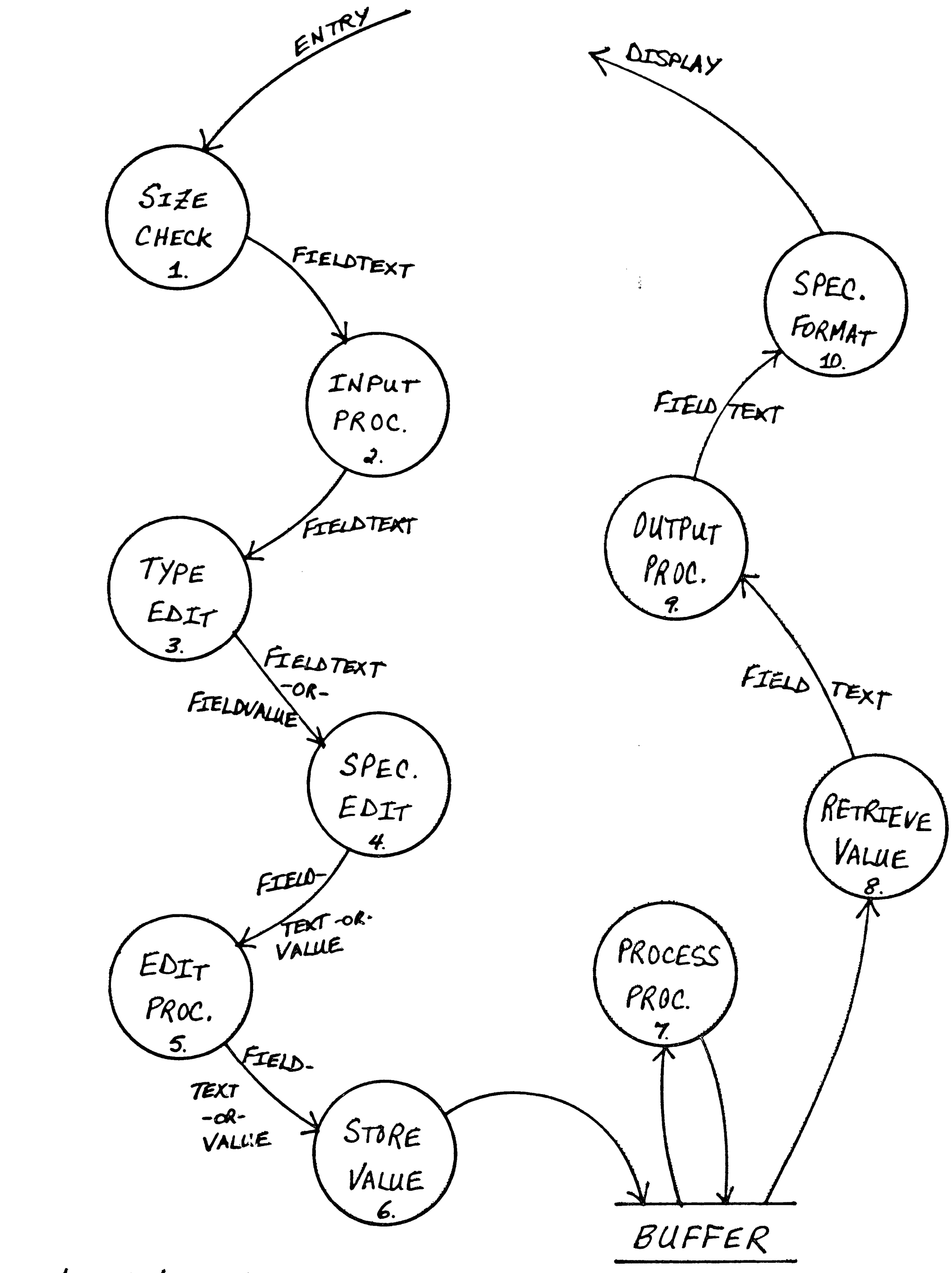
Figure 8



ACTIONs and Procedures Flow

ROBINSON, WALLACE & COMPANY: Page 6

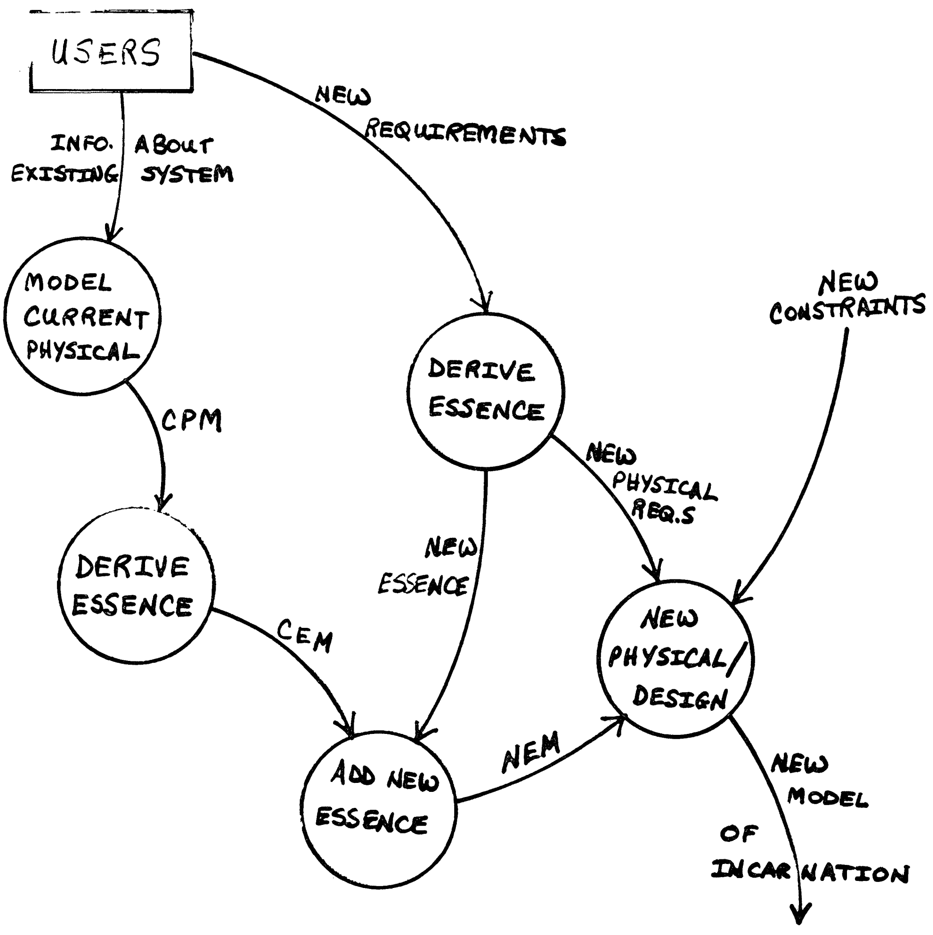
Figure 9



Data Entry Flow

Figure 10

Techniques --> Strategy Overview



Why do we care about system essence ? ...

3090. Techniques for Developing Device Independent Graphics Software

Peter Neuhaus
Peripheral Product Specialist
Hewlett Packard Company
19447 Pruneridge Ave
Cupertino, California 95014

Abstract

Occasionally the requirements for displaying data graphically go beyond the capabilities of existing software packages. Consequently, it becomes necessary to begin the unpleasant task of developing custom code. Since this typically involves a significant investment of resources, it is important that the software be written so as to maximize its longevity and maintainability. To achieve these goals, it is necessary to understand the basic principles of graphic device independence and the concept of the Virtual Device Interface.

This paper will discuss these topics along with a discussion of how available graphics software tools can be used to assist in the development of device independent graphics software.

Background

In the early 1970s, the computer graphics industry realized that it needed to standardize some of the methods used in developing graphics software. The resulting conventions made it possible to create graphics in one environment (computer) and transport them to another with a minimum of recoding. To date, only a few standards have been established but others are under investigation. The Graphics Kernal System (GKS) has been adopted by the International Standards Organization and is being used extensively throughout Europe while the Siggragh CORE system, proposed in 1979, has not gained much acceptance. The debate continues but GKS seem to be pulling ahead.

Regardless of whether or not one chooses to follow a strict standard, considerable improvements can be made in the writing of graphics software by following a few simple guidelines.

Frequently, companies plan to use only the specific graphics output devices that they already own, for example a HP7550 plotter or perhaps a non-HP graphics terminal. To support these devices, the specific instructions required by the devices would be scattered throughout the application program (see figure 1). The result would be very efficient but would necessitate excessive modifications if new or additional output devices were acquired at a later date.

The First Step

Device independence is nothing new to the professional programmer. Common functions such as cursor control are often modularized into separate subroutines (device drivers) that can be easily modified or replaced to accommodate new output devices that require different "escape sequences" for their proper operation. When it was necessary to drive more than one output device, a duplicate set of subroutines is written for each device (see figure 2). In addition, if more than one device might be used simultaneously, it is necessary for subroutines with indentical functions to have different names, such as LINE1, for drawing a line on device 1, or LINE2 for device 2. At this level, device independence was still not achieved since the LINE1 and LINE2 calls must be embedded in the application program.

Step Two

By inserting another level between the application program and the device drivers, the interface between the application program and the outside world is standardized. If this new level, perhaps a commercially available GKS package, contains a function that allows the application program to select which output device should be used, if is possible to remove the references to LINE1 and LINE2 and substitute a call to the new LINE function in the GKS package (see figure 3). At this point, true device independence has been achieved since new devices can be supported without modifying the application program as long as someone writes a device driver for the new device. However, creating these new drivers can consume enormous amounts of programming effort because each device is unique in that it requires specific nonstandard "escape sequences" to perform a given task.

VDI - The Last Step

The graphics industry is attempting to standardize the hardware instructions required by graphic output devices through a concept called the Virtual Device Interface (now often called the Computer Graphics Interface). If all graphic devices understood the same commands, the need for device drivers would be eliminated (see figure 4). Essentially the device drivers would be implemented within the device's firmware. However, until the VDI concept becomes commonplace, it is necessary to employ the basic concepts of device independence when writing graphics applications. Several alternatives are possible.

Ways to be Independent

The most straightforward solution would be to obtain a graphics software library either from the computer manufacturer or from an independent third party. Such packages include an number of device drivers for the most popular graphic devices. The disadvantage to this solution becomes evident if it is necessary to change host computers at a later date. Even switching between computer lines offered by the same manufacturer can cause significant problems. Therefore, when shopping for this type of software product, it is important to investigate the possibility of moving the product between systems. Packages written in standard languages such as Fortran or Pascal help simplify portability. But even standard languages often don't port well.

The ability to move to another CPU may sound like something that wouldn't be done too often, but as desktop computers become as powerful as typical multi-user systems, many applications will be moved to smaller workstations. It's much like the user who feels he needs only 50 megabytes of disc storage, orders 100MB even though he knows it will never be needed, then runs out of disc space six months later. Applications and technologies change continuously. Investing the extra resources to implement a flexible solution often pays high dividends at a future date.

Sharing Graphics Data

Frequently graphic databases created on one system need to be processed on another. To address this need, a standard format for exchanging databases, called the International Graphics Exchange System (IGES), has been established and is currently supported by a number of graphics packages. A similar newer standard, the Virtual Device Metafile (VDM), performs much the same functions. By simply using the IGES or VDM device driver, an application can store the resulting image or object description onto a transportable media such a magnetic tape which can then be read by another IGES/VDM compatible system. Applications written in a device independent manner are able to utilize this useful feature.

Summary

The trade-offs involved in the decision to standardize the development of computer graphics software deals mainly with short term verses long term benefits. Projects that seem to be "one shot" programs may not appear to necessitate the features of device independence. But often the programs are modified and used again, possibly for another "one shot" application. In general, establishing standards or guidelines in a programming environment leads to increases in productivity. The slight performance degradation created by the overhead of a graphics subroutine library can be offset by the ever decreasing costs of computer hardware. Once standards have been implemented, applications can be developed faster since it becomes unnecessary to reinvent the wheel for each new project. In addition, program maintenance is simplified since each programmer understands the basic strategies used by his fellow graphics programmers. Overall, the need to be device independent will become increasingly important as the number and capabilities of systems and graphic devices expand.

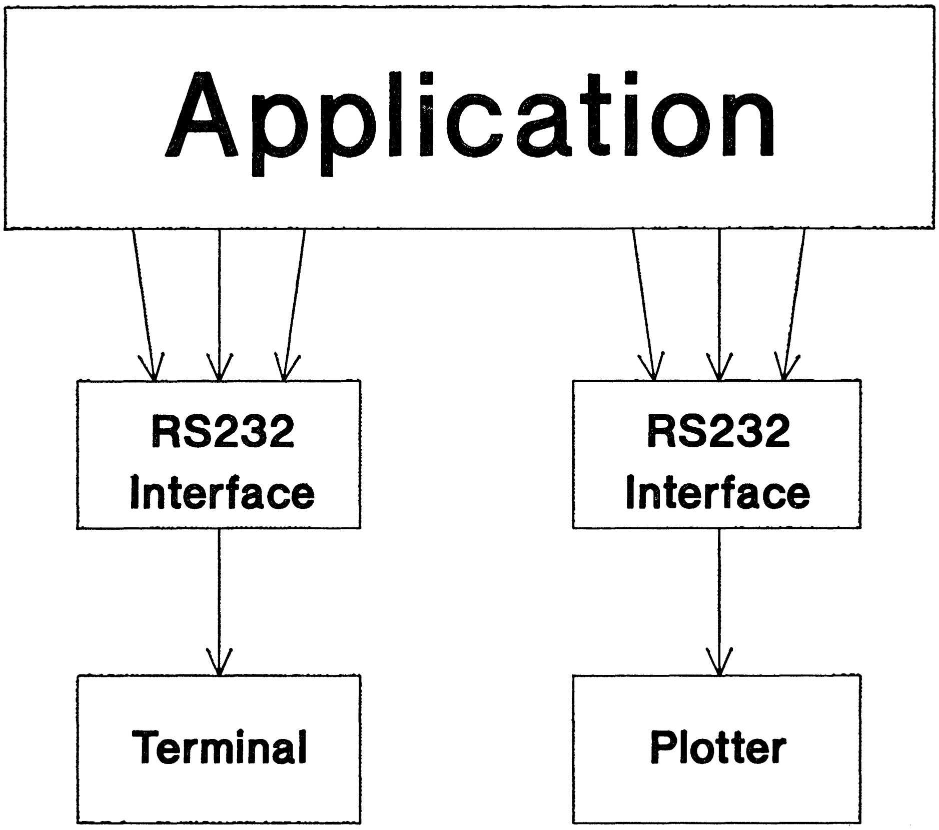

Application
RS232
Interface
RS232
Interface
Terminal
Plotter

Figure 1

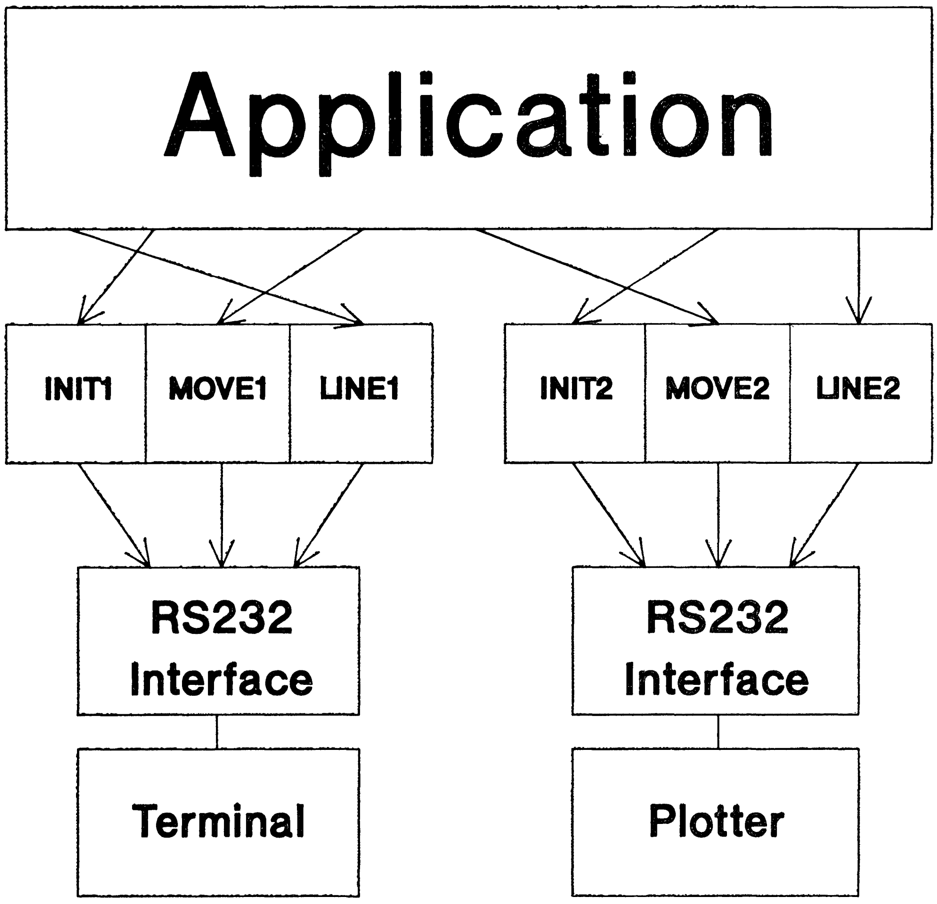

Application
INIT1
MOVE1
LINE1
INIT2
MOVE2
LINE2
RS232 Interface
RS232 Interface
Terminal
Plotter

Figure 2

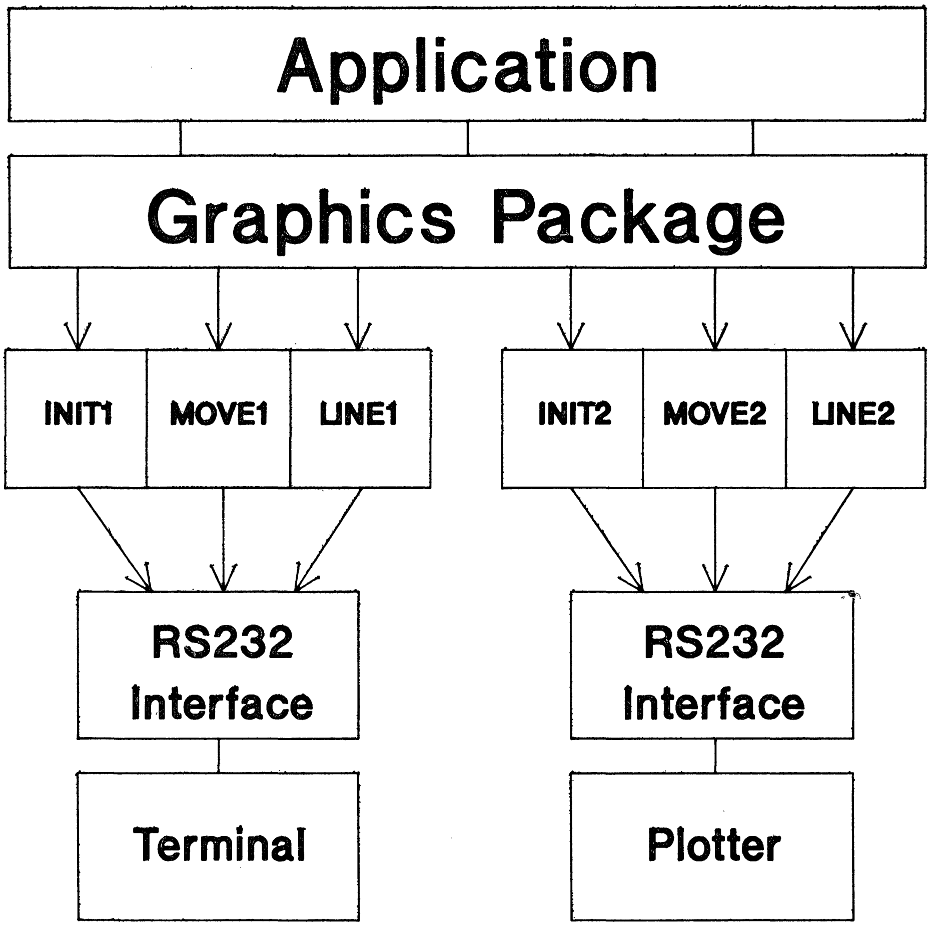

Application
Graphics Package
INIT1
MOVE1
LINE1
INIT2
MOVE2
LINE2
RS232 Interface
RS232 Interface
Terminal
Plotter

Figure 3

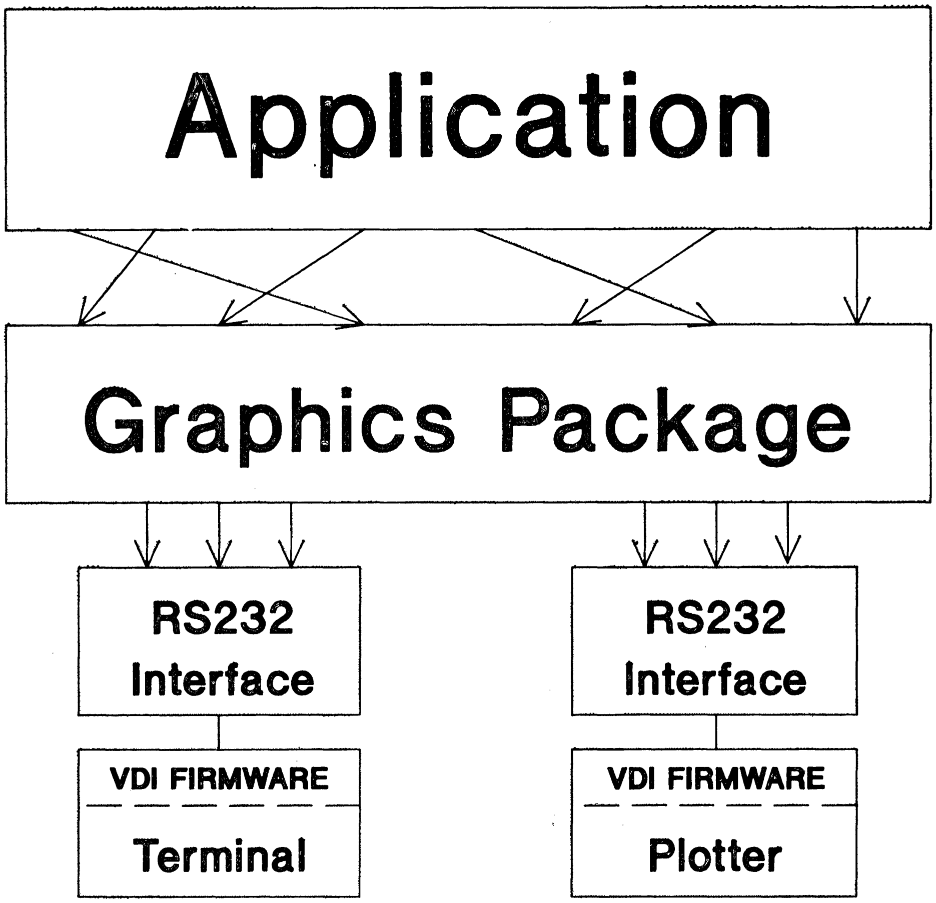

Application
Graphics Package
RS232 Interface
VDI FIRMWARE
Terminal
RS232 Interface
VDI FIRMWARE
Plotter

Figure 4

3091. Spool File Recovery - Without a Warmstart

Dwight P. Stewart
Hewlett-Packard
North American Response Center
Atlanta, Georgia

One of the facts that every system manager learns very early in his association with the HP3000 is that if he can't warmstart his system, his print files are gone forever. the purpose of this paper is to provide two alternatives to the lost spoolfile. Neither of the methods presented here are particularly mind-bending, but they do require some familiarity with the data structures involved with spooling, and some confidence in the use of Sadutil and Debug. The first part of the paper will provide some background on the data structures and the remainder will describe the actual techniques involved in rescuing the lost spool file. The step by step descriptions of each method of spool file recovery have been placed in appendices for future reference. (Experienced users feel free to skip to the juicy part, the rest of us will catch up later.)

The creature that we are working with here is a spoolfile. Its native habitat is the disc drive - as a matter of fact, any disc drive with device class "SPOOL" configured on it. Spool files can roam anywhere on the disc, there isn't any predefined reserved area where they are forced to stay. (This allows us to only use space for spooling when we need it.) Two major things distinguish a spool file from its tamer relative - the permanent disc file. First, the spool file does not have an entry in the system file directory. This is what causes spool files to be lost in the first place, their directory is rebuilt on cool and cold starts. The second distinguishing feature of a spool file is that its file label has a file name of blanks and a device class of SPOOL. Yes, spool files have file labels just like real files. The layout is identical to that of a permanent disc file, but the file name and device class give it away.

How does the system keep track of spool files if they aren't in the system directory? Easy, it uses a table. In this case, we have two tables, the Input Device Directory (IDD) for job streams and card reader input, and the Output Device Directory (ODD) for print files and spooled output. The IDD and ODD, (collectively known as the XDD) reside both on disc and in memory. The memory resident tables are the official ones, the disc copies are there so that people can warmstart and still have spool files. While the two tables are almost identical, let's focus attention on the output side of things and save the input spool files for later.

Like most MPE tables, the ODD has some extra goodies crammed into it along with the information we need. It has outfences for each output device, as well as the system outfence, and the spoolfile number of the next spool file to be created. (Note: here's a way to reset the next spool file number without doing a coolstart - just change the right word of the ODD in Debug).

A general layout of the ODD is as follows: one eight-word entry with global information such as table size, system outfence, and next spoolfile number. Next are four word header entries, one for each output device. The header entries are followed by 32-word subentries which exist for each output spool file and contain the disc address of the spool file label. The header entries and subentries are similar to Image master and detail records. Each header has pointers to the head and tail of a linked list of subentries. The subentries are linked to each other in only one direction, but have direct links back to the headers. The extra links can come in handy when trying to check yourself when using Sadutil.

Speaking of Sadutil, all this information about the ODD is pretty useless unless we can find it on disc. Guess what - another table! This one is called the Cold Load Information Table - sector 28 on the system disc (ldev 1). Sector 28 is one of the most useful portions of disc in data recovery situations because it has pointers to most of the disc resident data structures. The disc addresses of the IDD and ODD are here, as well as pointers to the volume table, which is our next topic.

The volume table keeps track of what volumes are mounted on each disc in the system domain. This is important because disc addresses on the 3000 consist of a volume number and a sector address. Spool file addresses being no exception, we have to look at the volume table to see which logical device our file is on. Luckily, most systems are installed with the volumes in the same sequence as the disc device numbers, so volume 1 and drive 1 are usually the same device. In cases where the volume table is trashed, we can use brute force to look at the same sector address on every drive until a given file is found. Just to complicate things, the disc addresses of the IDD and ODD do not have a volume number, the tables being assumed to be on the system disc.

Now that the basics have been covered, let's look at spool file recovery method #1 - disc recovery. This method requires either certain knowledge that the system will not do a recover lost disc space when booting, or at least an adventuresome spirit. This is because when using this method, all we do is dump the ODD and then recreate it after the system has been restarted. At no time do the spool files leave the disc. They are perfectly safe there as long as a recover lost disc space doesn't occur. Remember - space taken up by spool files isn't returned when you do a cool or cold start, you have to do a recover lost disc to get it back!

If the system went down with a disc error or a recover lost disc space is forthcoming, recovery method #2 becomes attractive - recovery to tape. Dumping the spoolfiles to tape is slower and less clever than method #1, but has the advantage of having the spool files safe from whatever trials and tribulations may be occurring on disc. With this method we also have to save the ODD and recreate it, but we also fool Sadutil into saving our spool files. We do this be finding each spool file label and putting a file name into the first four words of the label. We then instruct Sadutil to Save the file by address and it blithely puts the file on tape. (The file name is actually not a requirement of Sadutil, but of Recover, since the file has to be put back on the system somehow.) Once we have used Recover2/Recover5 to retrieve the file from the tape, we still have a bit of work to do. We find the disc address of the recovered file by doing a Listf,-1 and then create an ODD subentry using the old subentry and the new disc address. Now we have a potentially hazardous situation, a file which exists in both the ODD and the system directory. The file can be printed or examined in Spook, but don't try to purge it! Instead, after it has been copied or printed or Output to tape, purge the system file directory entry using Dirpur, and do a cool or coldstart just to tidy things up a bit.

Both methods are fairly time-consuming and probably shouldn't be attempted by a poor typist. In most situations, one can probably rerun a report more quickly/painlessly than recovering the spool file. However, for those really precious spool files, these methods may be life-savers.

Appendix A - Spool File Recovery Method #1

1. Load Sadutil and configure.

2. Dump sector 28 on logical device 1.

3. Words 34 and 35 (decimal) are the ODD disc address. The first eight bits are zeros, logical device 1 being assumed. The last eight bits are the high order end of a 24-bit sector address. If they are not zero, multiply by two and add to the first digit of word 35. Thus %000052 %175234 becomes sector %5375234. Remember, Sadutil defaults to decimal addresses unless specified otherwise.

4. Dump the first sector of the ODD.

5. Look at the right hand eight bits of word 0 to see how many sectors of the table are in use.

6. Dump the rest of the ODD.

7. Determine whether the files should be saved to tape. If the system went down with a disc problem or a recover lost disc space is pending, play it safe and use method #2, otherwise continue with this one.

8. Start the system. If a recover lost disc space occurred, or you had to do a reload, forget it. The spool files are gone forever.

9. Use Debug to look at the ODD in memory. Pointers to it are in the Data Segment Table entry 56, but all you need do is a DDA 56,100.

10. Most of the header entries that were recreated by the system will be identical to your Sadutil listing, all you need to do is modify the links for the devices which had spool files.

11. Re-enter each of the subentries related to spool files which are to be saved. (This is the time-consuming part, each entry is 32 words long). 12. Modify the forward links (word %35) of each subentry to point to the first word of the next subentry in this spool class.

13. Modify the links in the headers (words 2 and 3) to point to the first and last subentries on the list.

14. Gotcha: Bounds violation in Debug. You have tried to enter more spool files than the ODD currently has space for. Either create a large number of spool files and then purge them to force the table to be larger (it never shrinks), or enter as many subentries as you can and make several passes.

15. Do a showout, pat yourself on the back and print your spool files.

16. Congratulations!

Appendix B - Spool File Recovery Method #2

1. Load Sadutil and configure discs and tape.

2. Dump sector 28 of the system disc.

3. Dump the ODD using words 34 and 35 (decimal) of sector 28 as the address.

4. Word 2 of the ODD is an offset to the subentries. Words 22 and 23 (decimal) of each subentry are the disc address of the spool file label. The first eight bits of word 22 are the volume number, the last eight bits are the high order end of a 24-bit sector address. If the last eight bits are not zero, multiply by two and add to the first digit of word 23. Thus %000452 %115234 becomes volume 1, sector %5315234. Remember, Sadutil defaults to decimal addresses unless instructed otherwise.

5. Dump the file label. If it doesn't look like a spool file label, then the volume table may be in a different sequence and you will have to look at the other disc drives to see which one has your file.

6. Modify the file label and insert a sensible filename in the first four words. You may have to manually convert Ascii to octal if you don't have some artificial aid handy, Sadutil won't do it.

7. Use the Sadutil Save command to save the files on tape. Specify each file by disc device and address, it won't be able to find them by name.

8. Restart the system.

9. Use Recover2/Recover5 to put the spool files back on the system as permanent files.

10. Use Debug to resinsert each file back back into the ODD. Do this by re-entering the ODD subentry for the file and inserting the new disc address. You can get the disc address by doing a listf,-1 on the file name. The ODD is DST 46. The disc copy of the ODD need never be touched, the system will update that for you.

11. Fixup any header and subentry links as necessary.

12. Print the files but don't delete them or let the system delete them.

13. Use Dirpur to purge the system directory entries for each file.

14. Do a coolstart with recover lost disc space to get the ODD cleaned up and the disc space returned.

15. Congratulations!!