Proceedings

MAILED FROM AUSTIN TX POST MARK 669119 JUNE 15 82

TABLE OF CONTENTS

AUTHOR INDEX

Overview of Optimizing (On-Line and Batch)

Robert M. Green
Robelle Consulting Ltd.

SUMMARY

The performance of many HP3000 installations can often be improved significantly. There are general principles for delivering better response time to on-line users, and other principles to speed execution of production batch jobs. As long as users continue to consumer the extra horsepower of new HP3000 models by loading them with new applications, there will continue to be a need for optimizing knowledge and tools. And, if interest rates remain at current levels, many managers may not be able to upgrade to faster computers as soon as they would like.

CONTENTS

Copyright 1982, All rights reserved.
Permission is granted to reprint this document (but NOT for profit), provided that copyright notice is given.
This document was prepared with QGALLEY, a text formatter distributed with software to all Robelle customers.

SECTION 1
HOW TO IMPROVE ON-LINE RESPONSE TIME

I have identified five general principles which help in optimizing the performance of on-line programs:

- Make each disc access count.
- Maximize the value of each "transaction."
- Minimize the run-time program "size."
- Avoid constant demands for execution.
- Optimize for the common events.

On a systems programming project, such as a data entry package or a text editor, you should be able to apply all five of these principles with good results. That is because systems software usually deals with MPE directly and most of the sources of slow response are under your control. Applications software, on the other hand, usually depends heavily upon data management sub-systems such as IMAGE and V/3000. The optimizing principles proposed here may not be as easy to apply when so many of the causes of slow response are beyond your control. However, there are still many ways in which you can apply the guidelines to application systems (monitoring program size, designing your database and laying out your CRT screens). Relying upon standard software not only increases your programmer productivity, it also provides an unexpected bonus: any improvements that the vendor makes in the data management tools will immediately improve the efficiency of your entire application system, with no re-programming or explicit "optimizing" on your part.

I. A. Make Each Disc Access Acount

Disc accesses are the most critical resource on the HP3000. The system is capable of performing about 30 disc transfers per second, and they must be shared among many competing "consumers." (This can increase to 58 per second under the best circumstances, and can degrade to 24 per second when randomly accessing a large file.) MPE IV can double the maximum disc throughput for multi-spindle systems by doing "look-ahead" seeks, but only for the Series II/Series III, not the Series 30/33/44.

The available disc accesses will be "spent" on several tasks:

- Virtual memory management (i.e., swapping).
- MPE housekeeping (logon, logoff, program load, etc.).
- Lineprinter spooling.
- Accesses to disc files and databases by user programs (the final payoff).

If the disc accesses are used up by overhead operations, there will not be sufficient left to provide quick response to on-line user transactions. Some examples of operations that consume disc accesses on the HP3000 are:

- Increasing the number of keys in a detail dataset, thus causing IMAGE to access an extra master dataset on each DBPUT. Also, making a field a key value means that a DBDELETE/DBPUT is required to change it (which is 10 times slower than a DBUPDATE).
- Increasing the program data stack by 5000 words, thus causing the MPE memory manager to perform extra, swapping disc accesses to find room in memory for the larger stack.
- Improperly segmenting the code of an active program, causing many absence traps to the memory manager to bring the code segments into main memory.
- Constantly logging on and off to switch accounts.
- Defining a database with a BLOCKMAX value of 2000 words, thus limiting IMAGE to about 13 data buffers in the extra data segment that is shared by all users of that database. With such a small number of buffers, there can be frequent buffer "thrashing." This effectively eliminates the benefits of record buffering for all users of the database, and greatly increases disc accessing.

Much of the remainder of this document is devoted to methods of "saving the precious resource — disc accesses."

I. B. Maximize the Value of Each "Transaction"

This principle used to read, "Maximize the Value of Each Terminal Read," but I have generalized it to "transaction" to take into account the prevalence of V/3000, DS, MTS and other "communications" tools. In the terms of MPE IV, a "transaction" begins when the user hits the 'return' key (or Enter) and ends when the user can type input characters again. This includes the time needed to read the fields from the terminal (or from another HP3000), to validate them, perform database lookups and updates, format and print the results, and issue the next "read" request.

Each time a program reads from the terminal, MPE suspends it and may swap it out of memory. When the operator hits the 'return' key, the input operation is terminated, and MPE must dispatch the user process again. If MPE has overlaid parts of the process, they must be swapped back into main memory again. Due to the overhead needed to dispatch a process, a process should get as much work done as possible before it suspends for the next terminal input.

The simplest way to program data entry applications is to prompt for and accept only one field of data at a time. This is also the least efficient way to do it. Since there is an unpredictable "pause" every time the user hits 'return' (depending upon the system load at the moment), consistently fast response cannot be guaranteed. The resulting delays are irritating to operators. They can never work up any input speed, because they never know when the computer is ready for the next input line. If response time and throughput are the only considerations, it is always preferable to keep the operator typing as long as possible before hitting the 'return' key. Multiple transactions per line should be allowed, with suitable separators, and multiple lines without a 'return' should be allowed. If you are using V/3000, the same principles applies: each high-volume transaction should be self-contained on a single form, rather than spread out over several different forms.

I. C. Minimize the Run-Time Program "Size"

The HP3000 is an ideal machine for optimizing because of the many hardware features available at run-time to minimize the effective size of the program. Even large application systems can be organized to consume only a small amount of main memory at any one time. Each executing process on the HP3000 consists of a single data segment called the "stack," several extra data segments for system storage, such as file buffers, and up to 63 code segments. All segments (code and data) are variable-length and can be swapped between disc and main memory.

Program code which is not logically segmented makes it harder for the memory manager to do its job, causing disc accesses to be used for unnecessary swaps. Proper code segmentation is a complex topic (more like an art than a science), but here is a simplified training course: write modular code; don't segment until you have 4000 words of code; isolate modules that seldom run; isolate modules that often run; aim for 4000 words per segment, and group modules by "time" rather than "function;" if you reach 63 segments, increase segment size, but keep active segments smaller than inactive ones.

Although every process is always executing in some code segment, the code segment does not belong to the process, because a single copy of the code is used by all processes that need it. Since code is shared, it does not increase as the number of users running a given program increases. Most of your optmizing should be directed to the data areas (which are duplicated for each user). A 3000 can provide good response to more terminals if most data segments are kept to a modest size (5000 to 10,000 words). To keep stacks small, declare

most data variables "local" to each module (DYNAMIC in COBOL), and only use "global" storage (the mainline) for buffers and control values needed by all modules. Dynamic local storage is allocated on the top of the stack when the subroutine is entered, and is released automatically when the subroutine is left. This means that if the main program calls three large subroutines in succession, they all reuse the same space in the stack. The stack need only be large enough for the deepest nesting situation. By inserting explicit calls to the ZSIZE intrinsic, you can further reduce the average stack size of your program.

You can also minimize stack size by ensuring that constant data items (such as error messages and screen displays) are stored in code segments rather than in the data stack. Since constants are never modified, there is no logical reason that they should reside permanently in the data stack. By moving them to the code segment, one copy of them can be shared by all users running the program. In SPL, this is done by including =PB in a local array declaration or MOVEing a literal string into a buffer. In COBOL, constants can be moved to the code segment by DISPLAYing literal strings in place of declared data items. In FORTRAN, both FORMAT statements and DISPLAYed literals are stored in the code.

A frequently overlooked component of program "size" is the effect of calls to system subroutines (IMAGE, V/3000, etc.). These routines execute on the caller's stack, and the work they do is "charged" to the caller. In many simple on-line applications (dataset maintenance program, for example), 90% of the program's time and over 50% of the stack space will be controlled by IMAGE and V/3000. You should be aware of the likely impact of the calls that you make. Do you know how many disc accesses a particular call to DBPUT is going to consume? As an example of how ignoring the "extended size" of a program can impact response time, consider the following case:

An application with many functions can be implemented with one of two different strategies. The first, and simplest, strategy is to code the functions as separate programs and RUN them via a UDC (or CREATE them as son processes from a MENU program). Each function opens the databases (and forms-file, etc.) when you RUN it, and closes them before stopping.

The second strategy is to code each function as a subprogram that is passed in the previously opened databases (and forms-file, etc.) as a parameter from a mainline driver program. If the application requires frequent movement from function to function (performing only a few transactions in each function), the "process" strategy will be up to 100 times slower than the "subprogram" strategy. The resources required to RUN the programs, open the databases, close the databases, and perform other "overhead" operations will completely swamp the resources needed to perform the actual transactions.

I. D. Avoid Constant Demands for Execution

The HP3000 is a multi-programming, virtual-memory machine that depends for its effectiveness on a suitable mix of processes to execute. The physical size of code and data segments is only one factor in this "mix." The "size" of a program is not just the sum of its segment sizes; it is the product that results from multiplying physical size by the frequency and duration of demands for memory residence (i.e., how often, and for how long, the program executes). A given 3000 can support many more terminals if each one executes for one second every 30 seconds, rather than 60 seconds every two minutes. Each additional terminal that demands continuous execution (in high priority) makes it harder for MPE to respond quickly to the other terminals.

Here are some examples of the kind of operation that can destroy response time, if performed in high priority:

- EDIT/3000, a GATHER ALL of a 3000-line source file.
- V/3000, forms-file compiles done on four terminals at once.
- QUERY, a serial read of 100,000 records (or any application program that must read an entire dataset, because the required access path is not provided in the database).
- SORT, a sort of 50,000 records.
- COBOL, compiles done on four terminals at once.

You should first try to find a way to avoid these operations entirely. (Can you use QEDIT instead of EDIT/3000? Would a new search item in a dataset eliminate many serial searches, or could you use SUPRTOOL to reduce the search time? Are you compiling programs just to get a clean listing?)

After you have eliminated all of the "bad" operations that you can, the remainder should be banished to batch jobs that execute in lower priority (this works better in MPE IV than III). Since jobs can be "streamed" dynamically by programs, the on-line user can still request the high-overhead operations, but the system fulfills the request when it has the time. The major advantage of batch jobs is that they allow you to control the number of "bad" tasks that can run concurrently (set the JOB LIMIT to 1 for best terminal response).

I. E. Optimize for the Common Events

In any application where there is a large variation between the minimum and maximum load that a transaction can create, the program should be optimized around the most common size of transaction. If a program consists of 20 on-line functions, it is likely that four of them will be most frequently used. If so, your efforts should be directed toward optimizing these four functions; the other functions can be left as is. Because the HP3000 has code segmentation and dynamic stack

allocation, it is possible for an efficient program to contain many inefficient modules, as long as these modules are seldom invoked.

Since MPE will be executing a great deal of the time, you should become competent at general system tuning. Learn to use TUNER, IOSTAT, and SYSINFO (and the new :TUNE command in MPE IV). Any improvement in the efficiency of the MPE "kernel" will improve the response time of all users.

You do not have infinite people-resources for optimizing, so you must focus your attention on the factors that will actually make a difference. There is no point in optimizing a program that is seldom run. The MPE logging facility collects a number of useful statistics that can be used to identify the commonly accessed programs and files on your system. Learn to use the contributed programs FILERPT and LOGDB (Orlando Swap). If you are using IMAGE transaction logging, the DBAUDIT/Robelle program will give you transaction totals by database, dataset, program, and user (total puts, deletes, updates, and opens). Such statistics help in isolating areas of concern.

You can optimize application programs around the average chain length for detail dataset paths (the contributed program DBLOADNG will give you this information). Suppose you need to process chains of entries from an IMAGE dataset. If your program only provides data buffers for a single entry, you will have to re-read each entry on the chain each time you need it (extra disc I/O!!). Or, if you provide room for the maximum chain length, the data stack will be larger than needed most of the time (the maximum chain length is often much larger than the average). The larger data stack may cause the system to overload, eliminating the benefits of keeping the records in your stack. You should provide space in the stack for slightly more than the average number of entries expected. This will optimize for the common event.

SECTION II
ON-LINE OPTIMIZING EXAMPLE: QEDIT

QEDIT is a text editor for the HP3000 that was developed by Robelle Consulting Ltd. The primary objective of QEDIT is to provide the fastest editing with the minimum system load. Other objectives include conservation of disc space, similarity to EDIT/3000 in command syntax, ability to recover the workfile following a system crash or program abort, and increased programmer productivity.

QEDIT is an alternative to a hardware upgrade for users who are doing program development on the same HP3000 that they are trying to use for on-line production. Every optimizing paper in recent years by an HP performance specialist has recommended avoiding EDIT/3000. They usually recommend the "textfile-masterfile" approach to program development. (You do not actually edit your source program; instead, you create a small "textfile" containing only the changes to your "masterfile," then merge the two files together at compile-time). QEDIT allows you to have "real" editing on your HP3000, with less overhead than the "textfile masterfile" method, and still give good response time to your end-user terminals.

II. A. QEDIT and "Disc Accesses"

In order to reduce disc accesses, QEDIT eliminates the overheads of the TEXT, KEEP and GATHER ALL commands of EDIT/3000. These three operations have the most drastic impact upon the response time of the other users. QEDIT attacks the problem of KEEPs by providing an interface library that fools the HP compilers into thinking that a QEDIT workfile is really a "card image" file. As a result, it is never necessary to KEEP a workfile before compiling it. Since KEEPs are rarely used, most TEXTs are eliminated. The LIST command was given the ability to display any file (e.g., /LIST DBRPT1.SOURCE), so that a TEXT would not be required just to look at a file. TEXT is only needed when you want to make a backup or duplicate copy of an existing file. Since most users choose to maintain their source code in QEDIT workfiles (they use less disc space), the TEXTing of workfiles is optimized (by using NOBUF, multi-record access) to be four to seven times faster than a normal TEXT of a card-image file. The GATHER ALL operation is slow because it makes a copy of the entire workfile in another file. QEDIT renumbers up to 12 times faster by doing without the file copy.

Disc accesses during interactive editing (add, delete, change, etc.) are minimized by packing as many contiguous lines as possible into each disc block. Leading and trailing blanks are removed from lines to save space. The resulting workfile is seldom over 50% of the size of a normal KEEP file, or 25% of the size of an EDIT/3000 K-file (workfile). Most QEDIT users maintain their source programs in workfile form, since this saves disc space, simplifies operations (there need be only one copy of each version of a source program), and provides optimum on-line performance.

QEDIT always accesses its workfile in NOBUF mode, and buffers all new lines in the data stack until a block is full before writing to the disc. Wherever possible in the coding of QEDIT, unnecessary disc transfers have been eliminated. For example, the workfile maintains only forward direction linkage pointers, which reduce the amount of disc I/O substantially. Results of a logging test show that reducing the size of the workfile and eliminating the need for TEXT/KEEP reduce disc accesses and CPU time by 70-90%.

II. B. QEDIT and "Transaction Value"

Like EDIT/3000, QEDIT allows either a single command per line (/ADD), or several commands on a line, separated by semi-colons (/LIST 5/10;M 6;D 5). The principle of maximizing transaction value has been applied with good results to the MODIFY command. In

EDIT/3000, several interactions may be needed to modify a line to your satisfaction. QEDIT allows you to perform as many character edits as you like on each transaction; many users can perform all of their changes in a single pass. For complex character editing, such as diagrams, version 3.0 of QEDIT will provide "visual" editing in block-mode.

II. C. QEDIT and "Program Size"

QEDIT is a comletely new program, written in highly structured and modular SPL. The code is carefully segmented, based on the knowledge of which SPL procedures are used together and most frequently. Only two code segments need be resident for basic editing, and the most common function (adding new lines) can be accomplished with only a single code segment present.

QEDIT uses a modest data stack (3200 words) and no extra data segments. The stack expands for certain commands (especially the MPE :HELP command), but QEDIT contracts it back to a normal size after these infrequent commands are done. All error messages are contained in the code, isolated in a separate code segment that need not be resident if you make no errors.

Use of CPU time is th eother dimension to program "size." QEDIT is written in efficient SPL and consumes only a small amount of CPU time (compared with the COBOL compiler, or even EDIT/3000). Because QEDIT does its own internal blocking and deblocking of records, it can reduce the CPU time used in the ile system by opening files with NOBUF/MR access.

II. D. QEDIT and "Constant Demands"

Most QEDIT commands are so fast that they are over before a serious strain has been placed on the host machine. For example, a 2000-line source program can be searched for a string in four seconds. For those operations that still are too much load, QEDIT provides the ability to switch priority subqueues dynamically. In fact, the system manager can dictate a maximum priority for compiles and other operations that cause heavy system load.

II. E. QEDIT and "Common Events"

The design of QEDIT is based on the fact that program editing is not completely random. When a programmer changes line 250, he is more likely to require access to lines 245 through 265 next, than to lines 670 through 710. This observation dictated the design of the indexing scheme for the QEDIT workfile. There are many examples of optimizing for the most common events in QEDIT:

- Each block of a QEDIT workfile holds a "screenful" of lines, with leading and trailing blanks eliminated.
- QEDIT has built-in commands to compile, PREP and RUN (since these functions are frequently used by programmers).
- QEDIT has a fast /SET RENUM command (it can renumber 600 lines per second), instead of a slow GATHER command.
- QEDIT can TEXT a workfile much faster than a KEEP file (since most text will end up in QEDIT workfiles).
- QEDIT can "undo" the DELETE command (because programmers are always deleting the wrong lines).

II. F. Results of Applying the Principles to QEDIT

In less than seven seconds, QEDIT can text 1000 lines, renumber them, and search for a string. Commands are 80% to 1200% faster than EDIT/3000, program size is cut in half, and disc I/O and CPU time are reduced by up to 90%. There are now more than 350 computers with QEDIT installed, in all parts of the world. Recently, we asked the QEDIT users what they would tell another user about QEDIT. Here are some of their answers:

> "If he's doing program development, he needs QEDIT." (Gerald Lewis, Applied Analysis, Inc.)
>
> "Would not live without it. $INCLUDEs in FORTRAN; one file or dataset per include-file." (Larry Simonsen, Valtek, Inc.)
>
> "Fantastic product." (Lewis Patterson, Birmingham-Southern College)
>
> "Buy it. The productivity advantages are tremendous and don't cost anything in machine load. The disc savings in a large (13 programmers) shop will pay for it." (Jim Dowling, Bose Corp.)
>
> "It's great. We usually get into QEDIT and just stay there for a whole session. Compiles and PREPs are very easy. I really like FIND, LIST, and BEFORE commands. QEDIT is very fast. It is great for programmers." (Larry Van Sickle, Cole & Van Sickle)
>
> "It's a tremendous tool and should be used by any medium-sized shop. I use it to produce an index of all source or job streams for an account." (Vaughn Daines, Deseret Mutual Benefit Assoc.)
>
> "QEDIT is the best editor I've used on the market. It makes a programmer extremely efficient and productive. In rewriting an existing system completely, the on-line compile, flexible commands, and savings of disc space all contributed to bringing the system up very rapidly." (Glenn Yokoshima, HP Corvallis)
>
> "Excellent product. Increases programmer productivity dramatically (morale too!)." (David T. Black, The John Henry Company)
>
> "FAST, convenient. No need to TEXT and KEEP. Somewhat dangerous for novice, because changes are made directly. [It worked well for us in] conversion of SPSS, BMDP, and other statistical packages to the

HP3000." (Khursh Ahmed, McMaster University)

"If you are writing a lot of programs, you should get QEDIT. It is much easier than EDITOR for this purpose. Program source files demand complex editing capabilities, which QEDIT has. I shudder to think of having to work on a 4000-statement SPL source using EDITOR rather than QEDIT." (Bud Beamguard, Merchandising Methods)

"Excellent product. Anyone using the HP editor more than 6 times per day (or more than 1 hour/day average) should not be without QEDIT!" (T. Larson, N. J. McAllister and Associates Ltd.)

"Easier to use than HP editor and much more efficient. I do not have to leave QEDIT to RUN, PREP." (Myron Murray, Northwest Nazarene College)

"Takes a great load off the mind (i.e., the "electronic brain"). There have been occasions when heavy editing would have killed our system if we had been using EDITOR." (Mike Millard, Okanagan Helicopters Ltd.)

"Very good product — works well in development environment. Compilation of source programs without leaving QEDIT is very nice for debugging." (David Edmunds, Quasar Systems Ltd.)

"Use it. It is so much better than HP editor that there is no comparison." (Ilmar Laasi, TXL Corp.)

"Fast text editor." (F. X. O'Sullivan, Foot-Joy, Inc.)

"In one word. Fantastic." (Tracy Koop, Systech, Inc.)

"Superb tool. Far better than EDIT/3000. Also, information about HP3000 that is supplied gratis is very useful." (James McDaniel, The UCS Group Ltd.)

"I would highly recommend it over EDIT/3000. In benchmarks and actual use, it has proven to be much less load on the computer. In a University environment, we have many students and faculty editing programs at one time. QEDIT allows us to run with a high session limit and still get decent batch turnaround." (Dan Abts, University of Wisconsin — La Crosse)

"QEDIT is an excellent product for the price, and is one of the easiest ways to increase programmer productivity. The LIST command has been invaluable for cross-referencing data items in COBOL source programs." (Mark Miller, Diversified Computer Systems of Colorado)

"Absolutely. QEDIT has allowed us to control the development of systems (requiring off-line compiles, audit trails for source modifications) while actually increasing programmer productivity." (Jean Robinson, Leaseway Information Systems, Inc.)

"Get it! It's great. Cheap at twice the price." (Willian Taylor, Aviation Power Supply, Inc.)

"QEDIT is THE ONLY text editor that you should use in a development environment." (Craig T. Hall, Info-tronic Systems, Inc.)

"Much better than HP's editor, well supported, well documented and continually improving. An excellent product. We activate QEDIT from our job file generator and activate SPOOK from QEDIT for editing and testing output and job streams." (Patrick Hurley, Port of Vancouver)

"Excellent — can do more than Editor, faster, and saves disc space. In searching for a specific literal, QEDIT finds them all in one command [e.g., LIST "literal"]." (Larry Penrod, Datafax Computer Services Ltd.)

"We could probably not operate if QEDIT were not available." (Winston Kriger, Houston Instruments)

"Buy it, or another computer (a second HP3000, of course)" (John Beckett, Southern Missionary College)

"Best software package I've bought for our shop." (James Runde, Furman University)

SECTION III
HOW TO INCREASE BATCH THROUGHPUT

By a "batch job" I mean a large, high-volume, long-running task, such as a month-end payroll or financial report. Why is there any problem with this type of task? Because the batch job is only a poor, neglected cousin of the on-line session. "On-line" is "with it," new, Silicon Valley, exciting; "batch" is old, ordinary, IBM, and boring. The best people and most of the development resources have been dedicated to improving the on-line attributes of the HP3000. The result is predictable: batch jobs are beginning to clog many HP3000 processors. The overnight jobs are not completing overnight and the month-end jobs seem never to complete.

The methods for maximizing the throughput of a single batch job are not the same as for maximizing the response time of a large number of on-line users. The biggest difference: for an on-line application, it is seldom economical to optimize CPU usage. There isn't enough repetition to amount to much CPU time. But, a batch process may repeat a given section of code 100,000 or a million times. CPU time matters.

I have identified five general principles for increasing batch throughput. Not surprisingly, they differ significantly from the principles used to improve on-line response time:

- Bypass Inefficient Code (CPU hogs).
- Transfer More Information Per Disc Access.
- Increase Program Size to Save Disc Accesses.

- Remove Structure to Save Unneeded Disc Accesses.
- Add Structure for Frequent Events.

For each optimizing principle, there are three different tactics you can apply, with three levels of complexity and cost:

- Changes in the Data Storage (simplest and cheapest, since no programming changes are needed).
- Simple Coding Changes (still inexpensive, since these are "mechanical" changes which do not require re-thinking of the entire application).
- Changes to the Application Logic (the most complex and expensive, since the entire application may have to be re-designed).

III. A. Bypass Inefficient Code (CPU hogs)

Elimination of inefficient code is the simplest way to produce big throughput improvements, assuming that you can find any code to eliminate that is inefficient (or more general-purpose than needed).

For a number of reasons, IMAGE is usually more efficient than KSAM as a data management method. If you don't need "indexed sequential" as your primary access method, convert from KSAM files to IMAGE datasets. Or, if you don't need "keyed" access to the data, convert all the way from a data management subsystem to an MPE flat file, and use sequential searches. The more powerful the data access method, the more CPU time is required to maintain it.

Bypassing inefficient code is simply a matter of recoding parts of programs to substitute an efficient alternative for an existing method that is known to have poor performance. For example, the MPE file system is CPU-bound when handling buffered files, so converting to NOBUF access will save considerable CPU time (you transfer blocks and handle your own records). In IMAGE, use the "*" or "@" field list instead of a list of field names. In COBOL, re-compile your COBOL68 programs with the COBOL-II compiler and they will run faster. The FORTRAN formatter is a notorious "CPU hog"; either bypass it completely or learn its secrets. The third-party software tool, APG/3000 (application profile generator), should be helpful in identifying the portions of an application where the CPU time is spent (APG was written by Kim Leeper of Wick Hill Associates). Once APG has identified the key section of code, you might want to recode it in SPL/3000 for maximum efficiency.

As is usually the case, the biggest improvements are obtained by re-evaluating the logic of the application. For example, you should periodically check the distribution of all reports to see if anyone is reading them. If not, don't run the job at all — that is an infinite performance gain.

III. B. Transfer More Information Per Disc Access

Besides CPU time, the other major limit on throughput is the access speed of the discs. One way to transfer more information per disc access is to build files with larger blocksizes. The "block" is the unit of physical transfer for the file. A larger blocksize means that you move more records per revolution of the disc. However, there is a trade-off: increased buffer space and impact on other users. In on-line applications, you usually want a small blocksize. Below, I will explain NOBUF/MR access, which is a technique that allows you to "have your cake and eat it, too!"

Another way to transfer more useful information per disc access is to ensure that the data is organized so the records that are usually required together are in the same disc block. Rick Bergquist's DBLOADNG program (contributed library) reports on the internal efficiency of IMAGE datasets. For example, if it shows that the work orders for a given part are randomly dispersed throughout a detail dataset (necessitating numerous disc accesses), you can ensure that they will be stored contiguously by doing a DBUNLOAD/DBLOAD (assuming that part number is the primary path into work orders). For master datasets, DBLOADNG shows you how often you can find a specific entry with only a single disc read (the ideal). If DBLOADNG shows that multiple disc reads are often needed for a certain dataset, you may be able to correct the situation by increasing the capacity of the dataset to a larger prime number or by changing the data type and/or internal structure of the key field.

Don't overlook the obvious either. If you can compress the size of an entry by using a more efficient data type (Z10 converted to J2 saves six bytes per field), you can pack more entries into each block and thus reduce the number of disc accesses to retrieve a specific entry.

You can often increase the "average information value" of each disc access by re-thinking your application. For example, suppose you must store transactions in a database in order to provide some daily reports, many monthly reports, a year-end report, and an occasional historical report covering several years. If you store all transactions in a single dataset, the daily jobs will probably take three hours to find, sort, and total 100 transactions. Why not put today's transactions in a separate dataset and transfer them to the monthly dataset after the daily jobs are run? When the monthly reports are completed, you can move the data to a yearly dataset, and so on. This is called "isolating data by frequency of access." The fewer records you have to search to find the ones you want, the more information you are retrieving per access.

It is theoretically possible to transfer more information per second by reducing the average time per disc access. Typically, you attempt to improve the "head locality" (i.e., keep the moving "heads" of each disc drive in the vicinity of the data that you will need next).

Although it is hard to prove, it does seem that using device classes to keep spooling on a different drive from databases, for example, does improve batch throughput. Under MPE IV, you can also spread "virtual memory" among several discs. The next "logical step" is to place masters and details on separate drives. However, in all tests that I have run with actual datasets and actual programs, there was no consistent difference in performance between having the datasets on the same drive or on different drives. The dynamics of disc accessing on the HP3000 are very complex. Unless you have the time to do a RELOAD afterwards, don't move files around; the moving process itself (:STORE and :RESTORE) may fragment the disc space and eliminate the potential benefit of spreading the files. Remember Green's Law: "The disc heads are never where you think they are."

You can also improve overall batch throughput by recovering wasted disc accesses. The disc drives revolve at a fixed speed, whether you access them or not. Any disc revolution that does not transfer useful data is wasted. Multiprogramming attempts to use these wasted accesses by maintaining a queue of waiting tasks. Unfortunately, maximum throughput under MPE III coincided with JOB LIMIT = ONE (no multiprogramming!). Under MPE IV, however, I have obtained a 25% decrease in elapsed time on the Series III by running two or three jobs concurrently. Try it.

III. C. Increase Program Size to Save Disc Accesses

In on-line optimizing, we are always trying to reduce the size of the program (code, data, and CPU usage), so as to allow the system to provide good response time to more users at once. In batch optimizing, we do not want better response time (we won't be running 36 batch jobs at a time, so we don't have to worry about mix); we want better throughput. Since most of the on-line tricks actually make the program slightly slower, we should avoid them. Batch tricks usually consist of trading off a larger program size for a faster elapsed time.

You can often save disc accesses by storing data in larger "chunks," keeping more data in memory at any time. Larger blocks will accomplish this, as will extra buffers. MPE file buffers can be increased above the default of two via :FILE, but doing so actually appears to degrade throughput. KSAM key-block buffers are increased via :FILE (:FILE xx;DEV=„yy :MNS where xx is the KSAM data file and yy is the number of key-block buffers), which will help for empty files (KSAM cannot deduce how many buffers it will need unless the B-tree already exists). IMAGE buffers are increased via the BUFFSPECS command of DBUTIL; this can be effective for a stand-alone batch job, but only if it works with a large number of blocks concurrently (i.e., puts and deletes to complex datasets with many paths).

Pierre Senant of COGELOG (the developer of ASK/3000) has an ingenious method for "increasing program size" dramatically. He has implemented "memory files." An entire file is copied in main memory and kept there. For a small file that is frequently accessed (e.g., a master dataset containing only a few edit codes that must be applied to many transactions), Pierre's method should save enormous numbers of disc accesses.

NOBUF access to files was mentioned above as a way to save CPU time. If you use NOBUF with MR access, you can save disc accesses also, but at the cost of a larger data stack. MR stands for "multi-record," and gives you the ability to transfer multiple blocks per access, instead of just one block. With a large enough buffer, you will reduce the number of disc accesses dramatically.

Since multi-block access is faster only if each block is an exact multiple of 128 words in length, you should always select a recordsize and blockfactor such that the resulting blocksize (recordsize times blockfactor) is evenly divisible by 128 words. The resulting blocksize need not be large; it need only be a multiple of 128 (i.e., 256, 384, 512, . . .). As I promised earlier, here is your way to have the best of both worlds. Build your files with 512-word blocks (i.e., 4 times 128, 8 times 64, 16 times 32) for on-line use, and redefine the blocksize to 8192 words in batch programs via NOBUF/MR access.

For a "stand-alone batch" job, you may as well set MAXDATA to 30,000 words. This allows sorts to complete with maximum speed and provides other opportunities for optimization. With a larger stack you can keep small master datasets in the stack (e.g., a table of transaction codes). When you have exhausted the 30,000 words of your data stack, there are always extra data segments, which can be thought of as "fast, small files."

Re-evaluate your view of the data. Databases are usually set up to make life easy for the on-line user (rightly). Their organization may not be optimum for batch processing. In order to provide numerous enquiry paths, a single word order may be scattered in pieces among seven different datasets, and may require up to 20 calls to DBFIND and DBGET for assembly. In a batch job, if you are going to have to re-assemble the same order many times, it may be more efficient to define a huge, temporary record for the entire order, assemble it once, and write it to a temporary file. Then you can sort the temporary-file record numbers in numerous ways, and retrieve an entire order with a single disc read whenever you need it. Of course, this wastes disc space (temporarily) and increases your program size.

III. D. Remove Structure to Save Unneeded Disc Accesses

"Structure" for data means organization, lack of randomness, and the ability to quickly find selected groups of records. It takes work to maintain a "structure," and the more structure there is, the more work (CPU time and disc accesses) it takes.

Study your data structures critically. Can you reduce

the number of keys in a record? A serial search may be the fastest way to get the data. Can you eliminate a sorted path? Overall, the application may be faster if you sort each chain in the stack after reading it from the dataset (Ken Lessey's SKIPPER package has this capability), but only if you don't use the COBOL SORT verb.

Another type of "structure" is consistency. IMAGE is a robust data management system because it writes all dirty data blocks back to the disc before terminating each intrinsic call. You can make IMAGE faster, but less robust, if you call DBCONTROL to defer disc writes (only after a backup). Another IMAGE idea: don't use DBDELETE during production batch jobs. Just flag deleted records with DBUPDATE and DBDELETE them later, when no one is waiting for any reports. When you can, use a DBUPDATE in place of DBDELETE and DBPUT.

For KSAM, if you are planning to sort the records after you retrieve them, use "chronological access" (FREADC) instead of default access (FREAD). Default KSAM access is via the primary key; KSAM must jump all over the disc to get the records for you in this sorted order, just so you can re-sort them in another order! Also for KSAM, try to keep only one key (no alternate keys), do not allow duplicates (much more complex), and avoid changing key values of records.

I am grateful to Alfredo Rego for pointing out a useful way to "eliminate structure" from IMAGE. When you are loading a large master dataset, use a Mode-8 DBGET prior to the DBPUT in order to find out if the new entry will be a primary entry or a secondary entry. Load only primaries on the first pass, then go back and load the secondaries on a second pass. This effectively turns off the IMAGE mechanism known as "migrating secondaries," which although essential, is time-consuming when filling an entire dataset.

III. E. Add Structure for Frequent Events

I saved this for last because it is one of the most powerful ideas. Batch tasks usually repeat certain key steps numerous times. Batch tasks have patterns of repetition in them. If you make that key step faster by adding structure to it, or re-structure the application so that "like-steps" are handled together, you can make the whole task faster. Extra structure (code complexity or data complexity) is justified in the most frequent operations of batch processing.

Check your data structures for patterns that you could capitalize on. For example, if you have a file of transactions to edit and post to the data base, could the task be made faster if the file were sorted by transaction type (only do validation of the transaction type when it changes) or by customer number (only validate the customer number against the database when it changes)?

Here are more examples of adding structure. If you sort by the primary key before loading a KSAM file, you can often cut the overall time in half. When erasing an IMAGE detail dataset, sort the record numbers by the key field that has the longest average chain length and delete the records in that order. When loading a detail dataset with long sorted chains, first sort by the key field and the sort field. In all of these examples, throughput is increased by adding code structure to match the structure of the data.

If you frequently require partial-key searches on IMAGE records, use an auxiliary KSAM file (or a sorted flat file and a binary search) to give you "indexed-sequential" access, rather than only serial access, to your IMAGE dataset. (Mark Trasko's IMSAM product enhances IMAGE by adding an indexed-sequential access method to the other access methods of IMAGE.)

If you have used many IMAGE calls to find a specific record, remember its record number. Then, when you need to update it, you can retrieve it quickly with a Mode 4 DBGET (directed read), instead of doing the expensive search all over again. If certain totals must be recalculated each month, why not re-design the database so that they are saved until needed again? If something takes work to calculate, check whether you will need it again.

The general principle is: look for patterns of repetition and add structure to match those patterns.

SECTION IV.
BATCH OPTIMIZING EXAMPLE: SUPRTOOL

SUPRTOOL is a utility program for the HP3000 that was developed by Robelle Consulting Ltd. The objectives of SUPRTOOL are to provide a single, consistent, fast tool for doing sequential tasks, whether in production batch processing, file maintenance, or ad hoc debugging. Example tasks that SUPRTOOL can handle are: copying files, extracting selected records from IMAGE datasets (and MPE files and KSAM files), sorting records that have been extracted, deleting records, and loading records into IMAGE datasets and KSAM files. SUPRTOOL can't do everything yet, but we are adding new capabilities to it regularly (the most recent enhancements are a LIST command to do formatted record dumps and an EXTRACT command to select fields from within records). SUPRTOOL embodies many of the batch optimizing ideas discussed in the previous section of this document.

IV. A. SUPRTOOL and "Bypassing Inefficient Code"

By doing NOBUF deblocking of records, SUPRTOOL saves enough CPU time to reduce the elapsed time of serial operations visibly. For MPE files, NOBUF is now fairly commonplace (although it still isn't the default mode in FCOPY — SUPRTOOL is 6 to 34 times faster in copying ordinary files). Where SUPRTOOL goes beyond ordinary tools is in extending NOBUF access to KSAM files (a non-trivial task) and to IMAGE datasets (very carefully). By making only a

few "large" calls to the FREAD intrinsic, instead of many "small" calls to DBGET (each of which must access two extra data segments, look up the dataset name in a hash table, re-check user access security, and then extract a single record), SUPRTOOL quickly cruises through even enormous datasets with only a minimal consumption of CPU time.

For example, here is a comparison of SUPRTOOL and QUERY, selecting records from a detail dataset containing 60,971 current entries which are spread throughout a capacity of 129,704 entries.

```
SUPRTOOL/Robelle
>BASE ACTIVE.DATA,5
>GET LNITEM
>IF ORD-QTY>10000
>XEQ
IN=60971. OUT=14479.
CPU-SEC=56. WALL-SEC=133.
```

```
QUERY/3000
>DEFINE
DATA-BASE =>>ACTIVE.DATA
>FIND LNITEM.ORD-QTY>10000
USING SERIAL READ
14479 ENTRIES QUALIFIED
(CPU-SEC=520. WALL-SEC=763.)
```

Notice that SUPRTOOL used 1/9th as much CPU time and 1/6th as much elapsed time. And, the QUERY FIND command only builds a file of record numbers; to print the 14,479 records, QUERY must retrieve each one from the dataset again. SUPRTOOL creates an output disc file containing the actual record images, not the record numbers. With suitable prompting, SUPRTOOL can do this task even faster (see below for the BUFFER command).

IV. B. SUPRTOOL and "Transferring More Information"

SUPRTOOL transfers more information per disc access by doing multi-block transfers between the disc and the data stack in main memory. If records are 32 words long and stored as four per block (for a blocksize of 128 words), reading multiple blocks can make a big difference. For 20,000 records, one block at a time requires 5000 disc accesses. Using a 4096-word buffer and reading 32 blocks at a time reduces the number of disc accesses to 157!

SUPRTOOL has an option (SET STAT,ON) that prints detailed statistics after each task, so that you can see how it was done and where the processing time was spent. For example, suppose you want a formatted dump in octal and ASCII of all the records from the file described above for the order "228878SU." Below are the commands and times for SUPRTOOL and FCOPY:

```
FCOPY/3000
>FROM=SUMMRY;TO=*SUPRLIST;SUBSET="228878SU",1;OCTAL;CHAR
EOF FOUND IN FROMFILE AFTER RECORD 19999
3 RECORDS PROCESSED *** 0 ERRORS
(CPU-SEC=78. WALL-SEC=114.)

SUPRTOOL/Robelle
>SET STAT,ON
>DEFINE A,1,8
>IN SUMMRY
>LIST
>IF A="228878SU"
XEQ
IN=20000. OUT=3. CPU-SEC=11. WALL-SEC=16.

     ** OVERALL TIMING **
CPU milliseconds:              10854
Elapsed milliseconds:          16254
     ** INPUT **
Input buffer (wds):            4096
Input record len (wds):        32
Input logical dev:             12
Input FREAD calls:             157
Input time (ms):               6304
Input records/block:           4
Input blocks/buffer:           32
```

Notice that SUPRTOOL was using its default buffer size of 4096 words. FCOPY had to make 5000 disc transfers, while SUPRTOOL only had to make 157. That is one of the reasons why SUPRTOOL finished in 1/7th the time and used 1/7th the CPU time.

IV. C. SUPRTOOL and "Increasing Program Size"

SUPRTOOL gets a great deal of its performance edge by doing its own deblocking: allocating a large buffer within its data stack, reading directly from the disc into the buffer, and extracting the records from the blocks manually. SUPRTOOL trades a larger program size for a faster elapsed time. But you don't need to stop with the 4096-word buffer that SUPRTOOL normally allocates. Using the BUFFER command, you can instruct SUPRTOOL to work with buffers of up to 14,336 words and observe the results with SET STAT,ON. Here is the same selective file-dump that took 16 seconds with 4096-word buffers, done with 8192-word buffers:

```
SUPRTOOL/Robelle
>BUFFER 8192
>IN SUMMRY
>LIST
>IF A="228878SU"
>XEQ
CPU-SEC=10. WALL-SEC=13.  [An additional savings of 3 seconds]
```

By combining SUPRTOOL with IMAGE, you can have small data blocks for on-line access and large data blocks for batch sequential access. Here is the same database extract as done above (in the QUERY vs. SUPRTOOL test). Instead of using 4096-word buffers, we will increase the buffer space to 14,336 words:

```
SUPRTOOL/Robelle
>BUFFER 14336
>BASE ACTIVE.DATA,5
>GET LNITEM
>IF ORD-QTY>10000
>XEQ
IN=60971. OUT=14479. CPU-SEC=46. WALL-SEC=104. [Saved 29 sec.]
```

IV. D. SUPRTOOL and "Removing Structure"

SUPRTOOL can optimize batch operations by "removing structure." NOBUF deblocking of MPE files and IMAGE datasets provides faster serial access by saving CPU time and reading larger chunks of data, but NOBUF deblocking of KSAM files does that and more: it also eliminates structure. When you read a KSAM file serially by default, the KSAM data management system does not return the records to you in "physical" sequence; it returns them to you "structured" by the primary key value, and this takes work — a lot of work.

KSAM must search through the primary B-tree to find the sequence of the key values, and must then retrieve the specific blocks that contain each records. Quite often, logically adjacent records may not be physically adjacent; in the worst case, each logical record requires at least one physical block read. The SUPRTOOL NOBUF access to KSAM files cuts through all of this and returns the raw records to you in physical order; the savings in time can be impressive and, if you are planning to sort the records anyway, there is no loss of function. SUPRTOOL only removes the structure that you were not going to use.

Another example of removing structure in SUPRTOOL is the SET DEFER,ON command. When used in conjunction with the PUT or DELETE commands, the DEFER option causes SUPRTOOL to put IMAGE into output-deferred mode (via a call to DBCONTROL). Normally, IMAGE maintains a consistent and robust "structure" in the database after every intrinsic call. If you are planning to make a large number of database changes and can afford to store the database to tape first, you may be able to cut the elapsed time in half (or more) by leaving the physical database in an inconsistent state after intrinsic calls. (DBCONTROL makes the database consistent again when you are done.)

Here is an example use of SUPRTOOL to find all work orders that are completed (status="X") and old (dated prior to June 1st, 1982), delete them from the dataset, sort them by customer number and work-order number, and write them to a new disc file. SET DEFER,ON is used to make the DELETE command faster:

```
SUPRTOOL/Robelle
>BASE FLOOR.DATA
>GET WORKORDER
>IF WO-STATUS="X" AND WO-DATE<820601
>DELETE
>SORT CUSTOMER-NUM;SORT WORKORDER-NUM
>OUTPUT WO8206
>SET DEFER,ON
>XEQ
```

Another way to look at SUPRTOOL is as follows: if a serial search is fast enough, you may not need to have an official IMAGE "path" in order to retrieve the records you need. On the Series III, SUPRTOOL selects

records at a rate of two seconds per 1000 sectors of data.

IV. E. SUPRTOOL and "Adding Structure"

SUPRTOOL can optimize batch tasks by "adding structure" to data. One way to add structure is to sort data. Experiments have shown that sorting records into key sequence can cut the time to load a large KSAM file in half. SUPRTOOL easily reorganizes existing KSAM files by extracting the good records, sorting them by the primary key field, erasing the KSAM file, and writing the sorted records back into it — all in one pass.

You can also add "structure" to raw data by defining a record structure for it (QUERY can access IMAGE entries because they have a structure defined by the schema). Normally, regular MPE files are not thought of as having the same kind of record structure as IMAGE datasets. Why is this so? Because you cannot access the fields of the file's records by name in tools such as FCOPY, even if the structure exists. In SUPRTOOL, you can.

If you use SUPRTOOL to archive old entries from IMAGE datasets to MPE disc or tape files, you can later do selective extracts, sorts, and formatted dumps on those MPE files, using exactly the same field names as you did when the entries were in the database. (In fact, you can even put selected records back into a temporary database with the same structure and run QUERY reports on them.) Here is how SUPRTOOL associates structure with raw MPE files:

```
SUPRTOOL/Robelle
>BASE FLOOR
>INPUT WO8206 = WORKORDER
>IF CUSTOMER-NUM="Z85626"
>LIST
>XEQ
```

```
[implied record structure!]
```

And, since SUPRTOOL has access to the IMAGE database that the entries originally came from, SUPRTOOL can still format the entries on the lineprinter with appropriate field names and data conversions (similar to REPORT ALL in QUERY).

IV. F. Results of Applying Batch Rules to SUPRTOOL

Just before completing this paper, we sent a questionnaire to the users of SUPRTOOL, asking them what they would tell other HP3000 sites about SUPRTOOL. Here are their replies:

"I always recommend SUPRTOOL with any new system. Without programming, I duplicated a master file from one application to another application. I set up a job stream to do this on a weekly basis (i.e., purge the old dataset entries and add the new dataset entries easily). SUPRTOOL creates files with different selection criteria to feed the same program." (Terry Warns, B P L Corp.)

"An essential package for efficient operation of a system. Most of our job streams include a SUPRTOOL function." (Vaughn Daines, Deseret Mutual Benefit Assoc.)

"Excellent. We had an application that serially dumped a dataset of 185,000 records (4 hours) and then sorted the 114-byte records in 6 hours (provided we had the disc space needed). We changed to SUPRTOOL with the OUTPUT NUM,KEY option and a modified program using DBGET mode 4 and maximum BUFFSPECS. The result was 4 hours altogether." (Bobby Borrameo, HP Japan)

"SUPRTOOL is an excellent utility for copying standard MPE files and databases very quickly . . . extracting and sorting records from a database (i.e., 40,000 records of 60,000), copying files across the DS line (much quicker than FCOPY), copying tape to disc and disc to tape." (Dave Bartlet, HP Canada)

"We couldn't operate without it. We are a heavy KSAM user and SUPRTOOL has cut our batch processing by at least 1/3." (Jim Bonner, MacMillan Bloedel Alabama)

"All sorts of marvelous things. [SUPRTOOL] is really nice (and fast) to copy a database for test pruposes or to make minor changes (instead of DBUNLOAD/LOAD) — even major changes, using a program to reformat the SUPRTOOL-created file." (Susan Healy, Mitchell Bros. Truck Lines)

"Just last night I told a friend that, after working with different sorts on IBM (DPD- and GSD-level machines), Burroughs sorts, and even HP sorts, SUPRTOOL is the best sort tool I have ever used." (Robert Apgood, Whitney-Fidalgo Seafoods)

"Get it. Runs much faster than SORT. Cheap at twice the cost." (Willian Taylor, Aviation Power Supply, Inc.)

"Fast and functional. SUPRTOOL is deeply embedded in our applications, most extracts are done with SUPRTOOL. Ad hoc inquiries [via SUPRTOOL], involving pattern matching on our customer file, extract the appropriate keys, which are then passed to the report program." (Patrick Hurley, Port of Vancouver)

"SUPRTOOL is a product which no shop that uses IMAGE and does batch report generation should be without. By changing certain reports to use SUPRTOOL instead of traditional selection techniques, a savings of 60% in CPU and wall time was obtained." (Vladimir Volokh, VSI/Aerospace Group)

"SUPRTOOL is a great timesaver when used with BASIC (or RPG) to modify IMAGE datasets and place them in another dataset or the same dataset." (John Denault, Datafax Computer Services, Inc.)

Thoughts Concerning How Secure Is Your System?

Ingenieurbüro Jörg Grössler
IJG, Berlin

WHAT DATA SECURITY MEANS

- To be able to rebuild the file system in case of a disaster
- To restrict access on various type of data.

STANDARD FILE BACKUP FACILITIES IN MPE

- Sysdump, Reload (based on magnetic tape)
- Store, Restore (tapes)
- User Logging (based on disc or tape)
- Private volumes (disc)

PROBLEMS WITH STANDARD FILE BACKUP

- Tape read error during RELOAD
  — system cannot be started
  — next action "must be RELOAD"
  measures:
  — change disc packs before RELOAD
  — RELOAD with "ACCOUNTS-only" then RESTORE the remaining files (very time consuming)
- Tape read error during RESTORE
  — all files stored behind error point cannot be restored
  measure:
  — use RESTORE or GETFILE2 program
- User logging causes system overhead
  measure:
  — consider special logging during program design

PROSPECTS FOR TAPE-BACKUP SYSTEM

- GETFILE-facility will be improved
- Special STORE-RESTORE system is considered (this possibility includes features like UPDATE and APPEND)

RESTRICTIONS IN DATA ACCESS

- Account-system (users, groups, accounts with different passwords)
- User capabilities (SM, PM, PH, etc.)
- Filenames with passwords
- Privileged files
- File access capabilities on user/group- and file-level
- RELEASE/SECURE-commands

SEVEN POSSIBLE WAYS TO CRACK THE SYSTEM

1. FIELD.SUPPORT
   measure:
   Password on SUPPORT-account
   Or remove SUPPORT-account from the system.
2. Jobs in PUB.SYS-group
   measure:
   Password on job-file or
   Put job into other SYS-group.
3. LISTUSER@.@;LP
   measure:
   Log-on-UDC or perform command
   Not in PUB.SYS-group.
4. Open all files of the system
   measure:
   Special analysis of system logging
5. Read terminal buffers (PM-capability needed)
   measure:
   Remove PM-capability
6. Reading tapes
   measure:
   Keep track of all tape-transactions also using system logging
7. FOPEN on terminals
   measure: ??
8. . . .

Private Volume Experiences

Bruce Wheeler
Accounting Systems Group Cupertino, California

ABSTRACT

The MIS group which supports the accounting function within the Computer Systems Division of Hewlett-Packard has utilized private volumes for over two and one half years. This presentation will discuss actual user experiences as related to the following areas.

1. *Strategy* — where and how to successfully utilize private volumes. A discussion of tapes, system domain discs, and private volumes for storage of files will be included.

2. *Backup* — A comparision between serial disc backup and tape backup

3. *System Integrity* — Enhancing system integrity and lowering exposure to catastrophic errors. Tips to minimize exposure to disc errors and reloads will be included.

4. *Performance* — Performace considerations and trade-offs. Actual measurements on various HP3000 machines will be reported.

5. *Operator Considerations* — Simplifying the operator's assignment and improving the reliability of your data center.

BACKGROUND

The Accounting Systems Group of CSY reports to the CSY Controller and handles all accounting data processing for CSY. Our role within the Accounting Department is to support and develop computerized accounting systems. In addition to support we have become heavily involved and dedicated to:

1. Testing new HP products — both hardware and software. This includes not only doing pre-release testing for functionality and reliability but also utilizing these products to develop our distributed environment.

2. Fully utilizing HP software and hardware to implement a "distributed" data processing environment, i.e., one in which the computing power is where the people and problems are. This includes addressing the problems of system security and operatorless-computers.

We currently have our applications spread across three HP3000 systems, a SERIES 44 and SERIES 40 and SERIES 64 as Alpha test sites, with a total of about 1100 Mb of disc storage (seven of our disc drives are Private Volumes). One 2619A does the printing for all machines (we use DS/3000 to copy spoolfiles from the Series 40 and 44 to the Series 64). We have one HP125 microcomputer in the department. Our systems group of 11 professionals supports an accounting department of 40 people.

INTRODUCTION

The central objective of utilizing private volumes has been to increase system reliability, maximize computer throughput and minimize total operating costs. The mannner in which these objectives have been obtained are described in detail below. However, to fully appreciate the potential of these objectives, a clear understanding of private volumes and serial discs must first be realized. A disc spindle configured as a private volume provides an independent disc domain complete with its own directory. This domain may be moved from computer to computer with the requirements that like disc drives (i.e., 7920's on each system) be available and the same account structure exists. As a serial disc, the spindle assumes the nature of a tape drive. The spindle may be configured as both a serial disc and private volume at the same time. The current status of the drive is then dynamically allocated depending on the disc label of the particular pack which is mounted. This pack would be either a private volume or a serial backup disc.

RELIABILITY

System reliability is enhanced for a number of reasons when utilizing private volumes. As a storage medium, our experience has shown fewer catastrophic read errors from disc as opposed to tape. Although tapes do provide for storage of multiple files, it is certainly less cumbersome to retrieve a single file from disc then tape. And discs provide for direct as well as sequential access. In addition, if a segment of a disc is unreadable, it is possible to salvage the undamaged data, flag the offending tracks through VINIT, and reuse the pack. Only the data in the unreadable area is lost. Another advantage of private volumes is their transportability between machines. By establishing selected groups and accounts that contain critical data for processing, it is possible to freely move disc packs between computers if the primary machine is down. And within the same machine, if a system domain drive becomes inoperative, the system pack may be moved to a private volume spindle, the unit number dial on the

disc drive changed, and you are running with a warmstart. A more subtle advantage to private volumes, is that the master volume contains its own directory of the files in its volume set. This directory is independent of the system directory. In the event of a system failure that requires a reload, simply turn the private volume disc off, and reload the system with no loss of data on the private volumes.

THROUGHPUT

Private volumes provide a means of maximizing the system through put by replacing large sequential files on tapes. First, I/O to a disc is generally faster than that of a tape. By creating groups on private volumes for your production jobs that run at night which normally required tapes, the elapsed run time will decrease. And by having the VMOUNT ON,AUTO parameter activated, there is no wait time as would occur when the operator finally replies to the tape request. By having the private volumes mounted at the start of your nights processing, your operator does not have to be present for the job's execution. The groups mentioned above could be off-loaded during the day and replaced with groups that contain your source files for program development.

COST MINIMIZATION

In this area, both private volumes and serial discs provide a benefit. As mentioned above, by replacing tape files with those on private volumes, the operator does not have to be present for the job's execution. This may provide the possiblity of expanding processing to a second or third shift without the requirement for additonal support personnel. In addition, since private volumes have a directory that is tied to the group/account structure, it is not possible to have the wrong pack mounted and a data file read into your database as could easily happened with an unlabeled tape. For further insurance, by building a dummy file on the pack at the beginning of a job's execution, if the pack is not mounted, the job can abort in a controlled manner. For example:

```
:PURGE CHKMOUNT.PRIVOL.ACCT
:BUILD CHKMOUNT.PRIVOL.ACCT
```

In the above example, these JCL statements would be included in the beginning of the job stream. If the private volume was not mounted, the BUILD statement would fail so that a restart would only require restreaming the job. Backup time is reduced and the procedures simplified by using serial discs. A comparison between 7970 tape drives and a 7925 serial disc backup showed that the 7925 took aprroximately two thirds the elapsed time of the 7970. In additon, since a 7925 will store over three tape reels of data, the operator intervention for tape replies are reduced by a factor of 3 to 1. The operator is now required to perform the one task of the serial disc mount and then is free to proceed with other activites for approximately 1/2 hour.

START UP

The System Supervisor manual provides a detailed description of private volumes and serial discs in sections 4:10-4:13 and I:1-I:24. Listed below is a short summary of what it takes to establish a private volume and some tips in the utilization of this feature.

1. Setup the configuration through SYSDUMP. It is a good idea to have the class established for the disc to be both PVDISC and SDISC. This provides greater flexibility.
2. Create the volume set/class
3. Use VINIT to format the pack. Remember to flag those tracks as defective that have been listed on the tag. If you don't have the tag, then print the information with VINIT before you format and init the pack.
4. First span the account and then the group to link the system directory with the directory on the private volume. It is necessary to span at the account level even if only selected groups within the account are on the private volume.
5. For automatic recognition of the pack being mounted, have VMOUNT ON,AUTO set when you restart the system. In dismounting a pack, first do a VSUSER to verify that the volume set is not currently being accessed. Then down the LDEV to prevent further access and insure that the pack can be dismounted by doing a DSTAT· and checking that the state is not DOWN,PND.

APPLICATION FITS

- Large temporary files as in sorts. Create a group called SPACE that is an empty disc pack. With a 7925, these provide 120 MB's of free space by just mounting the pack.
- Offset your large nightly batch processing files with the program development source files during the day.
- Large Databases. Some databases, such as those in accounting, are cyclical in nature such that they are initialized every month. By having each month be on a separate pack, prior months can be retained for review if necessary.
- Security sensitive databases such as payroll and accounts payable can be off-loaded from the system and physically secured.
- Redundancy databases and IMAGE log files can reside physically on separate disc drives.

System Resource Accounting: An Overview of Available Software

Wayne E. Holt
Director of Computer Services
Union College
Schenectady, New York

Amy J. Galpin
Project Analyst
Whitman College Computer Services
Walla Walla, Washington

INTRODUCTION

Far too often in the minicomputer environment, the concept of system resource accounting (frequently called "job accounting") is overlooked by upper management. Such machines are cheap in comparison to mainframes, and the incentive to closely monitor usage is marginal.

There inevitably comes a day, however when the cheap little machine must do expensive and important work for too many people, resulting in slower throughput and performance. And that is when upper management confronts the DP manager with the question "Say, who is using up the time? Run us a report that pinpoints the problems."

Most DP managers will have already experimented with some of the resource accounting software available through the Contributed Software Library. Few, however will have a well defined philosophy or methodology of resource accounting that is well supported by the proper software. Usually, upper management will have denied the requests to invest manpower into such an unnecessary system. In the worst cases, accounting needs will have been so overlooked, that when the DP manager rushes to test some of that library software, he will discover that the MPE logging facility hasn't been enabled! Logfiles, notorious for disc space consumption, might also have been quickly purged by the operator.

It is a premise of this paper that accurate and timely information regarding system resource usage is essential for data processing management. The HP3000 Contributed Software Library contains numerous programs and software packages to aid in the collection and evaluation of job accounting data. This paper will examine the available library software, summarizing the strengths and best usages for each. In addition, Whitman College will serve as the example in a case study illustrating the complimentary nature of using in-house developed software with externally acquired programs.

Only software available on Release 07 of the Contributed Software Library, or on the ORLANDO Swap Tape will be discussed. In addition, it should be noted that several organizations and vendors now have general resource accounting software available for sale. One can gain information about such software by reading the advertisements in Interact, or by asking the HP sales representative to check his software reference guide. It is our understanding that HPIUG will be offering such a guide sometime in 1982, as will several private parties. One should also note those software packages, in the Release 07 Guide, with an "F" by the page number of the corresponding index entry. This indicates that the software is available by contacting the vendor appearing in the abstract, although a fee is charged.

The term "system resource accounting" was chosen to title this paper because classic "job accounting" implies keeping records on job or session activities, including such information as start time, stop time, CPU usage, disc I/O counts, etc. This does not encompass the full spectrum of information available on MPE logging records, e.g., powerfail information and console messages. Furthermore, externally developed data such as manually maintained timesheets, although pertinent, is ignored.

The balance of this paper will be split into four sections, with an appendix following. Section I will discuss software that processes "special" MPE log records. Section II will cover the simpler series of programs that yield traditional job accounting information, while Section III will deal with more complex methodologies and software systems. Finally, Section IV is a case study of the approach Whitman College has taken to begin satisfying its system resource accounting needs.

In each of the three sections, the general purpose of the software will be described, its similarities or differences to other software will be discussed, and if appropriate, comments will be made concerning how to run the software. Finally, an asterisk occuring by the software package name indicates that sample results may be found in Appendix A. Before continuing, the reader might take time to review the summary of MPE logfile record types, located in the HP manual.[1] In order to facilitate your evaluation of the results, the same logfile, LOG2345.PUB.SYS, has been used in all software

runs. The following types of logging are enabled on our system:

Type of Logging producing	Record Type No.
Logging Enabled	0, 1
Job Initiation	2
Job Termination	3
Process Termination	4
File Close	5
System Shutdown	6
Power Fail	7
Spooling	8
I/O Error	11

Note that the console logging is disabled. We do not perform statistical analyses on this information and have found a hardcopy console log to be more useful in monitoring this "scene of action".[2]

SECTION I: SOFTWARE FOR "SPECIAL" MPE LOG RECORDS

Software which processes "special" log records, such as powerfails and console messages will be discussed in this section. Special software performing utility functions will also be discussed. Programs falling into this category tend to be standalone (with a few exceptions) and their operation is fairly straightforward. In most cases, it is advisable to examine the source code to ensure that the utility is applicable to your system's configuration. Modifications in such things as equated constants, often those that reflect logfile record size, as well as others, may be necessary to make the software run properly.

CLISTLOG[3]

This program provides a report of all console log records (type 15) in the MPE logfiles . The format of the report is in chronological sequence, using perhaps only 1/3 of the paper consumed in HP's LISTLOG2 report of type 15 records. No statistical analysis on the log records is performed. The utility is similar to JLISTLOG, SLISTLOG and LISTLOG2 in both operation and function. While JLISTLOG and SLISTLOG report on predetermined logfile record types, LISTLOG2 allows the User to specify the type of records desired at run time. All four of the utilities are capable of traversing across a range of logfiles. The User is prompted for the number range of LOG####.PUB.SYS to be searched. The User is also given the option of purging the logfiles after the search. This utility could be quite useful at a site where a hardcopy console log is not used and management wishes to peruse/review this realm of system activity at a later time. The User may direct output by equating the file CLOGLIST to the desired output device. Console logging MUST be enabled for this program to serve its purpose.

CONSLOG[4]

This contribution produces a report of those console log records occurring in MPE logfiles for a given date/time range. The User is not only able to select records by date, but also by defining a character string which must occur in the type 15 records. Output may be directed to a device other than $STDLIST, and the program is capable of building files on disc if a non-existent file is specified for output; input will also be accepted from a command file.

The program prompts the User for a starting date and time to be used as the beginning point of the search, as well as an output file and search string. The program is capable of continuing the search across logfile boundaries, up to the current logfile. The author suggests performing a: SWITCHLOG before running the program, if the User wishes to examine the current logfile (requires OP capability). As mentioned above, the console logging must have been enabled during system configuration to produce the type 15 records the program searches for.

This program would also be useful for installations in which a hardcopy console is not employed, or where management wishes to monitor the appearance of specific users, job/session names, etc. on the system. Because output may be directed to a disc file, the User may develop his own procedures to sort, reformat, or edit the output, according to his needs.

COSTPROG[5]*

This utility calculates the cost of data center services by considering the CPU seconds, connect minutes, and disc sector usage of a group. The report is broken down by account and group across the three elements listed above, and is similar in format to the listing produced by the MPE: REPORT command. The User is able to specify his own cost parameters.

The program does not read MPE logfiles, but instead reads a data file produced by previously issueing the: REPORT command, where output was directed to a disc file. The User is prompted for the cost factors, and can direct output by equating formal file designators to the desired device. The input file equation should be set before running the program.

The User is limited to producing figures only for those accounts he has the capablity to: REPORT on (to: REPORT on all accounts requires SM capability).

JLISTLOG[3]*

Belonging to the family of CLISTLOG and SLISTLOG, this program produces a report of all job initiation and job termination records (types 2 and 3) within a given range of logfiles. The listing is formatted in chronological sequence, and again, consumes approximately 1/3 of the paper consumed by a LISTLOG2 listing of the same records. A page break occurs with each new date.

The User is prompted for the starting and ending logfile numbers; the program looks for them in PUB.SYS. The User is also given the option of purging the logfiles after the search, and will be asked if he wishes to run the program again. Currently accessed logfiles are not available to the program. While LISTLOG2 requires SM capability, JLISTLOG does not.

The contributor recommends that the source code be examined, to ensure that the logfile record size of your installation coincides with that in the source code; modifications should be made before attempting to run the program.

LISTLOG2[6]*

While this utility does not appear on either the library release or swap tapes, it is an HP product universally available to HP3000 users, and seems appropriate for review. This MPE utility produces an ASCII listing of any number of logfile record types across a given range of logfiles. The report is chronologically ordered and record entries are separated by hyphenated break lines.

Operation of the utility is similar to that of the CLISTLOG family. Indeed, this utility is the general version after which the specialized CLISTLOG family is modeled. The User is prompted for which types of logfiles, if not all, he wishes to report. He must also specify the beginning and ending numbers of the logfiles he wants searched, and has the option of purging the specified logfiles after the search. The User is asked if he wishes to run the program again before its termination.

The program is versatile in that output may be directed to any file on any device (e.g., disc or mag tape file as well as line printer). The program is restricted to Users with SM capability.

LOGPURGE[7]

This utility purges a given range of logfiles LOG####.PUB.SYS. The User is prompted for the beginning and ending numbers of the logfile range. The logfile being currently accessed will not be purged.

The program is similar in function to PURGELOG of the DREEACTG software package.

PFAILIST[8]

This program scans logfiles within a given range, and prints the date and time of each logged powerfail. The User is asked to input the starting and ending logfile numbers.

This program could be especially useful to a site in which there is no hardcopy console log to record powerfail messages.

PORTSTAT[8]*

PORTSTAT will scan a given range of logfiles, performing statistical analysis to produce a report on the usage of various ports on the system. Total CPU seconds and connect minutes, as well as the average figure per job/session and standard deviation are broken out against the ldev number. A combined CPU sec/connect minute figure (presumably weighted) is also given. The report heading gives the date/time range of the logfiles scanned. The User is prompted for the logfile number range. The port number range is controlled by equated constants and should be tailored to your site's configuration. Output is to $STDLIST.

READLOG[9]

This utility will carry on a dialogue with the User, scanning a logfile for records selected according to the User criteria input. Logfile records may be sought out by criteria such as such as record type, ldev origination, date, and time range, or in combination. The program will also summarize the number of occurrences of each record type before terminating. By asking for an audit of the logfile, the first and last records will be displayed (handy for finding date/time range of logfile). The User may specify a new logfile to be scanned, without having to reiterate the criteria.

The program opens the logfiles as LOGXXXX.PUB, and therefore should be run in the SYS account. This program could be very useful as a "lead" in monitoring system activity. Only summarizations are performed by the program. While the program does recognize all logfile record types, it does not decode all data items to ASCII format.

SCANUSER[10]

This program produces a report of all activity for which a log record was produced relating to a particular or generic group of Users.

The program issues a prompt for the User name in question (or generic user.acct) , and then for a logfile number. The program is capable of handling up to 15 concurrent users, and is most informative when most logging functions are enabled.

SLISTLOG[3]*

This program is another member of the CLISTLOG family, its function being to seek out spool file close log records (type 8) within a given range of logfiles. The report produced is in chronological sequence, and formatted with uniform column headings which break out each type of data element occuring in the log record. A page break occurs upon each new date encountered.

The User is prompted for a logfile range to be searched, and is given the options of purging the logfiles, and/or running the program again.

SECTION II: "SIMPLE" SOFTWARE FOR JOB ACCOUNTING

This section will discuss the "simpler" software that can be used to derive job accounting information. The

software in this area is generally easy to use, and requires the least amount of preparation. Relatively little statistical analysis or summarization is performed on the data, and results tend to be of a highly detailed nature.

LOGDB[11]

Briefly summarizing, this software system is designed to read system logfiles, loading them into an IMAGE database. The structure of the database is one that allows for simple report generation via QUERY or application programs. Some summarizations are performed upon the data. It is loaded in nearly "raw" form to the database, with the conversion of some data elements to ASCII format. The system is also capable of "rationalization" which eliminates a good deal of redundant data.

The system, as it is available on the Orlando swap tape, includes "first time" jobstreams, intended to compile all source code and initially create the database. Daily procedures are also incorporated into several job streams which jointly serve to read the logfiles, load the database, and produce reports while performing any "housekeeping" necessary to accomplish this. The reports provided are generated by QUERY through the execution of several command files. While highly detailed in nature, the reports may serve well as skeletons by which a site can tailor its own reports.

In a little greater detail, the general structure of the database is as follows: Paths are defined by several automasters, however one manual
"job-head" master exists which holds information needed for several of the detail sets. There is a detail data set for each type of log record encountered in system logfiles, with these exceptions:

1. Console log records are written to a console log.
2. Job initiation and job termination records (types 2 and 3) are combined into one detail set. This is the JOB-INIT/TERM data set which also houses several count fields. The set has been designed to facilitate billing from one set.

The detail data sets are loaded on a one entry per log record basis, except for the JOB-INIT/TERM set mentioned above, which houses some summary fields, and the LOGICAL-MOUNT set which contains only one entry for each job or session, and holds a total field. Entries are also not created for certain types of file closes, although they might be added to I/O count fields.

No duplicate job/session numbers are allowed on the database, thus if the loading program encounters duplicate numbers within a group of logfiles, it assumes the most recently encountered as the current job/session. This problem can, for the most part, be avoided by processing logfiles on a daily basis rather than in large groups. The data set capacities are currently set to acommodate approximately four logfiles. This may be altered to your site's needs. Overall, the input to the system consists of log files. Output consists of a loaded database, a hardcopy console log (assuming console logging has been enabled), and an error listing to $STDLIST. An in flight processing summary report may also be output to a terminal by using a control-y interrupt; the last logfile to be processed, and the logfile being currently processed are displayed. The system also creates a few working files which ensure continuous processing of logfiles between daily runs.

SECTION III: "COMPLEX" SOFTWARE FOR JOB ACCOUNTING

The use of software appearing in this section is perhaps not as straightforward as that in the preceding section. Proper use of the software to yield meaningful results requires that a an accompanying methodology be developed and followed on a regular basis. These packages are capable of performing a greater amount of statistical analysis on the data accumulated, producing reports of a higher summary level. The packages generally also provide opportunities to produce highly detailed reports, depending at which phase of the process one finds oneself.

DREEACTG[12]*

This software system actually tracks system utilization in two manners, the first via the processing of MPE logfiles, producing job/session information, and the second recording disc storage utilization using data created by the: REPORT command. The two systems are independent, however they both consist of a series of daily procedures which accumulate information, with another series of periodic (monthly) procedures designed to summarize and present the data in various formats. The modular structure of the system allows a site to use the software in its entirety, or to utilize those portions of the package applicable.

The system is capable of a large amount of statistical analysis, producing highly detailed reports which accompany the daily data accumulation, as well as periodic summary reports which break out the data in several manners. The reports could be highly useful to an operations staff in monitoring system resources, to account managers and/or project leaders by informing them of system activity associated with their "domain," as well as to DP management in holding various cost centers accountable for system usage.

While the method of cost center assignment is specifically geared toward the account structure found at the contributor's site, this logic module is a self contained subroutine which could easily be altered to a site wishing to apply its own philosophy of cost center assignment.

Cost computations are performed using weighting factors held in an initialization subroutine, as well as a cost limiting factor; these factors can also be easily reviewed and modified by a site wishing to weight or limit computations in a different manner.

In a slightly more detailed consideration of the job/session processing portion of the package we see that the daily procedures involve two steps. The first is performed by the program ACUMLOG, which functions to summarize by job/session all activity accounted for in the MPE logfile under that job/session number. Summary files produced in the first step are then read by the program LOGRPT, which appends cost fields to the activies, producing a monthly summary file in the second step. It might be useful to note that only log record types 2,3,4,5,8 and 9 are considered. Only summary records for job/session numbers less than 1000 are processed by LOGRPT; this can also be altered for sites whose job/session numbers commonly surpass this limit. It might also be worthy to note that one must take care to manually ensure that summary files for different months are kept separate.

Reports produced include logfile summary reports, job/session detail reports and monthly reports, produced from the monthly summary files, are broken out by account and at the group level, and cost center level. Invoices may also be produced, broken out primarily by cost center, and at a secondary level by the account structure within.

The disc storage utilization portion of the package generates reports in a manner similar to those mentioned above. Data is obtained from directing the output of a: REPORT command to a disc file. The data thus obtained is then accumulated on a daily basis by the program ACUMDISC which creates a monthly master file after cost fields have been appended. The disc charge rate is hard coded into the program and can be easily changed.

Extended documentation of the system can be obtained. This outlines detailed operating procedures, most of which are incorporated into jobstreams.

This package is an example of the incorporation of another library contribution, ACUMLOG[7] into an in-house tailored job accounting system.

LOGUTIL[13]*

This user oriented and highly versatile software package is designed to serve four general functions. It facilitates the storage of logfiles in a randomly accessible format, it scans logfiles, selecting and displaying log record types chosen by the User, it summarizes various types of activity logged within the file, and it analyzes such summaries in terms of job/session activity, file activity, or device I/O errors. The program is versatile in its ability to accept input and output both to and from disc or tape, or in combination. This is controlled through file equations. Various report options are given the User within each generic type of report. Options include such items as detail level, sort-item, and rank item within sort item.

The package consists of the three programs LOGUTIL, LOGREPT and FILERPT; a data file is also required which reflects your site's configuration. LOGUTIL is the central program of the system, performing the storage, summary, and scanning functions, as well as the evaluation of I/O errors. The other two programs, LOGREPT and FILERPT, analyze the summary files produced by LOGUTIL, to produce the various job/session and file activity reports.

Briefly, the program LOGUTIL allows for the selection of three functions. Logfiles may be copied to tape in a consolidated fashion (multiple reels are supported), an audit review of the logfiles performed, or an I/O error analysis reported. Output depends upon the option selected and may include a "loaded" tape, a listing of the number of records in each logfile, starting and stop time, a listing or file of summaries for each job/session, summaries file, a file activity summary file, and a summary listing showing the number of occurances of each record type in the logfile. Logfiles may be optionally purged, if the User has SM capability.

The program LOGREPT analyzes the job/session summary file produced by LOGUTIL. The User is prompted for such selections as the listing device, the input file name, the report date range, whether to analyze by groups or users, any groups or users to be excluded from analysis, and the detail level of the report (long or short); the program can also provide account summaries.

The program FILERPT carries on a healthy dialogue with the User, in a triple nested loop fashion. The file activity report produced may be "viewed" by files accessed, by name or rank of access, and by User accessing the files. Likewise, the report may be presented primarily by Users accessing files, by name or rank of access, and with or without the files accessed being listed. There are several sort items from which the User may choose to "rank" output. Counts and totals are also given.

The contributor recommends that the source code of LOGUTIL and LOGREPT be examined and modified to handle your system's configuration.

The operation of the system is quite well documented.

SECTION IV:
A CASE STUDY OF WHITMAN'S SYSTEM RESOURCE ACCOUNTING

Since its beginning in mid-1977, the Computer Services organization has kept records of work performed for the various offices on campus. These records included some computer-generated information on machine and paper usage, as well as manually maintained records on human resource usage.

Frankly, upper management cared little about such records. Most resources were adequate, and User concerns centered around when they would "get their turn."The Computer Services Office used available software to occasionally monitor the system usage, and

correctly predicted the inevitable shortfall of computer resources. The software mentioned above included a rudimentary Manpower Accounting System, dating to mid-1978, created using student labor. This had been planned as one part of a larger Job Accounting system, however manpower was never made available to complete the task. Thus, the Center relied upon such packages as DREE to monitor actual computer usage. While these packages were more than adequate to get a measure of system activity, they did little to provide comprehensive evaluations of the overall impact of various User offices.

The onset of lack of resources forced a change in most everyone's thinking. In pursuing the creation of a five-year plan for computer usage on campus, the Computer Policy Committee recognized the need for usable, consistent data for planning. While the aforementioned Manpower Accounting reports were of help, most of the computer-generated information was simply not in a "digestible" form. This resulted in some justifiable criticism of the material presented in support of the proposed five-year plan. The supporting figures were primarily directed toward manpower usage, with only highly technical information available on machine utilization. The support and maintenance functions were not delineated from development functions, and at times, were aggregated with both the User and the Computer Center itself. No actual dollar figures tied back to real expenditures were presented. In general, the User was left with an incomplete picture of the amount and type of activity on the computer.

Development of a "diary" database, and supporting programs then ensued. The resulting system resource accounting system, called the DIARY, was designed to fulfill information needs in three areas. It accounts for manpower resources, computer resources, and material resources, such as paper. The three areas taken into account are made unique from each other in the level and type of data collected, as well as the collection methods used. The areas also have several common features, namely the resulting derived data and the philosophical approach used in deriving summary level data. Unique requirements are addressed by logic modules designed to meet those needs. The common requirements are fulfilled by the conversion of usage figures into standardized units, useful in analysis.

It would perhaps be best to briefly summarize the philosophy of the DIARY, and then proceed with a more detailed description of the software and methods used in its support.

We wished to present system resource accounting information in a manner which would correlate not only the types of resources being used with the application system receiving the benefits of such usage; it was necessary to indicate whether the resources were invested in production work within a system, maintenance of the system, or development of an entirely new system. Furthermore, the activities of the Computer Center staff needed to be represented in a way which delineated between the general overhead needed to maintain the organization, and services rendered accountable to specific offices. It was also necessary to separate usage figures generated by administrative functions from those generated by academic functions. While many shops might be able to keep accurate figures by strictly designating logon accounts to be used for specific purposes, we wished to gather more detailed data, in terms of computer resources. While Whitman College does not employ actual charge-back, the structure of the database preserves this as a viable alternative for the future.

The three logic modules of the DIARY are named by the type of information which they address. They are the Unit of Manpower (UOM), the Unit of Processing (UOP), and the Unit of Resources (UOR) modules. While the UOM is daily in orientation, the UOP and UOR modules summarize data by the month. The unique features of each module will be discussed after examining the set of common "unifying" codes which are derived from the various types of data encountered in each module. The authors acknowledge the fact that while the following codes presented are applicable to the specific information needs of Whitman College, they may not be entirely appropriate for shops in a different environment. It would be helpful in the discussion that follows, to examine the examples, in Appendix B, of how these codes are employed in our shop.

- Activity Area — denotes a general type of activity, (Production, Maintenance, Primary or Secondary Development. Primary development involves the creation of a new system; secondary development involves the addition of new programs or functions to an existing system.)
- Sector — makes a general distinction between administrative, academic and computer center functions
- Office — Designates a particular administrative office, or academic division. (Registrar, Admissions, Division of Social Sciences, etc.)
- System — an application system in which work is being performed. Each system is "assigned" to an office which is held accountable for the system. (Payroll, General Ledger, Class Grading, etc.) Each academic department also is assigned its own system code (Physics)
- Project (SR) — Project numbers are assigned to any production or maintenance work which is performed in response to a service request, and is performed by a computer center staff member. A project may affect several User systems, and the format of the number allows evaluation on either a project or system orientation.

The office code is functionally analogous to the cost center. Due to the heirarchical structure of the codes, those at the top may be derived simply by "climbing the

ladder." While this may appear redundant, the design of the system was partially dictated by the ad hoc inquiry tools available, such as QUERY[14] and QUIZ[15]. Efforts could then be concentrated upon creating systems to load the database, rather than in creating report programs.

Separate tables drive the software that tags and identifies the aforementioned codes to each job accounting record. Some of these tables require a minimal amount of manual maintenance. Such maintenance might be due to the creation of a new account, or to the installation of a new system. While the assignment of codes and cost centers may be completely by defaults, the tables allow for proper assignment of codes in exceptional cases.

The UOM records the actual hours worked within the computer shop by all staff. The UOM is the straight-time dollar/hour rate of an employee, multiplied by the hours reported. Whether the employee is of exempt or non-exempt status is ignored, resulting in a "weighted" charge for services depending upon the person providing it. All hours are designated to a system code, and an activity code; the activity code is more specific than the activity area, however can be mapped to the activity area after considering the system/activity code combination. A project number might also be optionally recorded, provided it is compatible with the activity code (e.g., a maintenance project code would not be compatible with a system undergoing primary development). Data in this module is handled in a fairly specific manner. Accountablilty goes even to the program number being worked on, where the program number takes the place of an activity code. Reports can be generated by employee, by system, by activity, and by project. Such detailed reports may not be of great interest to upper management, but are useful to staff members in visualizing where their efforts are spent. It is important to emphasize that unless staff members reports all hours worked, regardless of whether or not they are paid, the accuracy of the UOM module as a planning tool is considerably degraded.

The Unit of Processing module provides comprehensive information relating to machine utilization. The UOP is a derived figure making use of weighting factors built into the accounting processor obtained from the Department of Regional Economic Expansion (DREE). A portion of the DREE software is used in the first step towards loading the UOP leg of the DIARY. The DREE package is used to summarize raw data from the log-files, by session and by job, and to append cost fields to the activities represented. Because our philosophy in assigning cost centers is different from that of DREE, in-house developed software then performs the remaining steps in loading the database. The following paragraph briefly describes our method of assigning cost centers.

While DREE incorporates the cost center code into the User name, we have found that cost centers are not so "cut and dried" in our shop. It is relatively safe to assume that any work performed in an administrative User's account is production work, as well work performed in student or faculty accounts as being academic. However, accountability as to the software system being used is lost. The major problem is in tracking the type of work being performed by Computer Center staff members, which may be development, maintenance, or in support of User production. We have resorted to extensive use of standardized job/session names which vary according to the type of work the staff member intends to perform when logging on. The combination of job/session name, user name, and logon account is checked against the code assigning tables mentioned previously. From the tables, the cost center is derived. The structure of the job/session name is either of the form of a specific project number, or of the format "system code/general activity area." Thus we are able to account for work performed on the computer down to the system level; the type of work is inherent in the project number, or in the activity area, whichever is used. The UOM module also accounts for activity under project number, and activity codes are mapped to general activity areas, as mentioned before.

Using ad hoc inquiry methods, reports can be generated that delineate between production, development, or maintenance work; The amount of each type of work taking place within application systems can also be reported. And most certainly, reports may present in both text and graphical forms the comparative usage of processing power, as well as staff manpower, by administrators, faculty, and students.

The Unit of Resource (UOR) module provides data relating to paper usage. The UOR is a derived figure, relating to the print-line count obtained from MPE log-files, and summed by DREE software employed.

The DIARY database is diagrammed in Appendix B - I.1. Software should be run on a regular basis to summarize and transfer data between the sets. The emphasis is upon running timely detail reports, and then eliminating any accumulation that is unnecessary.

Unless Users know how much it costs to provide them services, it will be difficult to prioritize or separate actual needs from "wish lists." Cost values are maintained for UOM, UOR and UOP, although they are approximate until the close of the fiscal year. Such data relating actual expenses to Computer Center activities is a necessary planning tool for Users, and is helpful only if made available on a timely basis.

The investment in software for all of this is actually very modest. The "lions share" is for the load programs, and those that handle summarization. Of course, reports of greater precision will ultimately be developed. Because the system was targeted for ad hoc reporting, the software investment shall continue to be minor. The following is a summary of current programs

and their functions in the flow of information within the DIARY:

JA120 — UOM Staff Activity Report Input Screen

JA202 — Summarizes UOM-DTL, loading ACTIVITY-DTL

JA204 — Summarizes UOM-DTL and UOP-DTL, loading the SUMMARY-DTL

JA234 — Transfers DREE records to UOP-DTL, adding record heads

JA323 — UOM Monthly Manpower Report by Employee

JA325 — UOM Monthly Manpower Report by System

JA327 — UOM Monthly Manpower Report by Activity

JA329 — UOM Monthly Summary of Employee Hours

JA405 — UOP Monthly Summary of Machine Utilization by Sector

JA640 — SUMMARY-DTL Report of UOM, UOP, and UOR against activity area, within system, within office

In addition:

The PROJECT-MST, EMPLOYEE-MST, and BUDGET-DTL are manually maintained via DBENTRY[16].

CONCLUDING REMARKS

The classic concept of "job accounting" is inadequate to provide management with a proper understanding of the cost involved in providing total service to Users. Only total "system resource accounting," which includes manpower, equipment, and material resources can hope to provide the divergent types of data needed.

While few shops will find free or fee-charged software that adequately meets their needs, there is a wide variety available to begin with. Much of this is free to members of the Users Group. It is important that shop management recognize the need to gather such data, before confrontations with upper management prompt the need. Certainly, each shop will need to tailor any general purpose accounting software system to their own environment. Better to start early, for a large base of historical data is usually required to establish trends.

All of the above reinforces the need to plan early. The authors of this paper hope that the material and considerations presented will help you formulate the appropriate course of action for your shop.

REFERENCES

[1]Detailed information on logfiles may be found in the HP System Manager/System Supervisor Manual, Section VI.

[2]See "The Hardcopy Console: A Tool for Installation Management," by W.E. Holt, Montreux Proceedings, 1980.

[3]Contributed by Linford Hackman, Vydec, Incorporated. [4]Contributed by S.G. Joerger, Armament Systems Incorporated.

[5]Contributed by Bill Klages, DE Systems, Incorporated.

[6]An HP product; see the MPE System Utilities manual, Sect. IV, for operating instructions.

[7]An anonymous contribution.

[8]Contributed by Jon Falconer, Pacific Union College.

[9]Contributed by John A. Maus, Hewlett-Packard.

[10]On Orlando Swap tape, Bob Dunn programmer.

[11]Contributed by The Bose Corporation.

[12]Contributed by Serge Bazinet, Department of Regional Economic Expansion, Govt. of Canada.

[13]Original author was Gerry Wade, contributed by Brent J. Thompson, The Development Office, BYU, with some modifications.

[14]An HP product for on-line inquiry. See IMAGE and QUERY manuals.

[15]Produced by Quasar Systems Ltd.

[16]An IMAGE-VIEW interface program, contributed by Bruce Kau, Tours, Incorporated.

APPENDIX A

Table of Contents

```
:BUILD RRR;REC=-72,,,ASCII;DEV=DISC;DISC=1000
:FILE RRR,OLD;DEV=DISC
:REPORT @,LIB,*RRR
:FILE FTN01=RRR,OLD
:FILE FTN07;DEV=COMLP
:RUN COSTPROG;STACK=2000

ENTER COST PER CPU SEC IN DOLLARS ?0.001

ENTER COST PER CONNECT MINUTE IN DOLLARS ?0.01

ENTER COST PER SECTOR IN DOLLARS ?1.00

END OF PROGRAM
:FILE OUTFILE;DEV=COMLP
:RUN PSCREEN.UTIL.IRIS
LOCKWORD: PSCREEN,UTIL,IRIS?

**SCREEN CONTENTS @: WED, JAN  6, 1982,  11:08 AM
```

ACCOUNT	FILESPACE-SECTORS		CPU-SECONDS		CONNECT-MINUTES		DOLLAR
/GROUP	COUNT	LIMIT	COUNT	LIMIT	COUNT	LIMIT	CHARGES
LIB	16888	**	2555	**	2645	**	$15,917.00
/DATA	589	**	24	**	18	**	$589.20
/DOC	783	**	77	**	191	**	$784.99
/JOB	132	**	0	**	0	**	$132.00
/PUB	10244	**	2185	**	2374	**	$10,269.93
/SOURCE	5140	**	269	**	62	**	$5,140.89

```
   TIME      JOB#   ON/OFF      JLISTLOG A.00.00    DATE: FRI, DEC  4, 1981,  1:33 PM       LOGFILE: 2345

13:33:50:3  #J690     ON     FDDEV,NYHAGEN,WCCS,N1; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
13:40:00:2  #J691     ON     UFUPDTCA,BUSENTRY,ADMIN,BUSINESS; PRI=DS; CPU=5000; INPRI=8; OUTPRI=8; JIN=10; JLIST=17
13:41:00:9  #S2893    OFF                          CPU-SEC=43; ELAPSED-MIN=43; MAXPRI=0; NUM-CREATIONS=15
13:41:56:5  #J691     OFF                          CPU-SEC=7; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=3
13:42:51:9  #S2952    OFF                          CPU-SEC=33; ELAPSED-MIN=23; MAXPRI=0; NUM-CREATIONS=8
13:43:11:5  #S3098    ON     SANDERCR,STU84B,SANDERCR; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=53; JLIST=53
13:43:36:9  #S2683    OFF                          CPU-SEC=140; ELAPSED-MIN=90; MAXPRI=0; NUM-CREATIONS=48
13:43:55:5  #S3100    ON     LUTTGEJC,STU85B,LUTTGEJC; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=47; JLIST=47
13:44:10:8  #J692     ON     UFUPDTCA,BUSENTRY,ADMIN,BUSINESS; PRI=DS; CPU=5000; INPRI=8; OUTPRI=8; JIN=10; JLIST=17
13:45:08:2  #J690     OFF                          CPU-SEC=62; ELAPSED-MIN=12; MAXPRI=0; NUM-CREATIONS=4
13:45:10:1  #S2694    OFF                          CPU-SEC=18; ELAPSED-MIN=47; MAXPRI=0; NUM-CREATIONS=1
13:45:13:4  #J693     ON     FDDEV,KELSEY,WCCS,K1; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
13:45:35:1  #J692     OFF                          CPU-SEC=8; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=3
13:45:39:3  #J694     ON     GL1AJOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=17
13:47:25:6  #S3102    ON     MICHELSO,ADMIN,ALUMNI; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=57; JLIST=57
13:47:38:4  #J694     OFF                          CPU-SEC=15; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=7
13:49:26:2  #J695     ON     GL1AJOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=4; OUTPRI=0; JIN=10; JLIST=17
13:49:57:7  #S3108    ON     FDDEV,NYHAGEN,WCCS,NYHAGEN; PRI=CS; CPU=UNLIM;HIPRI; OUTPRI=0; JIN=27; JLIST=27
13:51:55:7  #S2877    OFF                          CPU-SEC=4; ELAPSED-MIN=123; MAXPRI=0; NUM-CREATIONS=1
13:52:09:4  #S3108    OFF                          CPU-SEC=4; ELAPSED-MIN=3; MAXPRI=0; NUM-CREATIONS=1
13:57:14:1  #S3113    ON     FDDEV,NYHAGEN,WCCS,N1; PRI=CS; CPU=UNLIM;HIPRI; OUTPRI=0; JIN=27; JLIST=27
13:52:41:9  #J695     OFF                          CPU-SEC=14; ELAPSED-MIN=4; MAXPRI=0; NUM-CREATIONS=7
13:52:45:4  #J696     ON     FDDEV,MANAGER,WCCS,COMMON; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
13:54:58:2  #J696     OFF                          CPU-SEC=14; ELAPSED-MIN=3; MAXPRI=0; NUM-CREATIONS=1
13:55:09:2  #J697     ON     FDDEV,MANAGER,FINDEV,PUB; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
13:56:13:9  #J697     OFF                          CPU-SEC=5; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=1
13:57:13:2  #S3113    OFF                          CPU-SEC=5; ELAPSED-MIN=6; MAXPRI=0; NUM-CREATIONS=1
13:57:19:7  #J698     ON     GL1AJOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=4; OUTPRI=0; JIN=10; JLIST=17
13:57:32:2  #S3124    ON     COTTREFM,STU84B,COTTREFM; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=44; JLIST=44
13:59:02:9  #S3126    ON     FDDEV,NYHAGEN,WCCS,N1; PRI=CS; CPU=UNLIM;HIPRI; OUTPRI=0; JIN=27; JLIST=27
14:01:22:5  #J698     OFF                          CPU-SEC=13; ELAPSED-MIN=5; MAXPRI=0; NUM-CREATIONS=7
14:01:27:6  #J699     ON     UFUPDTCA,BUSENTRY,ADMIN,BUSINESS; PRI=DS; CPU=5000; INPRI=8; OUTPRI=8; JIN=10; JLIST=17
14:02:50:7  #S3124    OFF                          CPU-SEC=11; ELAPSED-MIN=6; MAXPRI=0; NUM-CREATIONS=4
14:03:00:9  #J699     OFF                          CPU-SEC=7; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=3
14:03:14:0  #J700     ON     GL1AJOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=17
14:03:57:5  #J693     OFF                          CPU-SEC=125; ELAPSED-MIN=19; MAXPRI=0; NUM-CREATIONS=4
14:04:23:0  #S3126    OFF                          CPU-SEC=4; ELAPSED-MIN=6; MAXPRI=0; NUM-CREATIONS=1
14:04:34:2  #S3132    ON     FDDEV,MANAGER,WCCS,BATCH; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=27; JLIST=27
14:05:51:6  #J701     ON     UFDSJENT,BUSENTRY,ADMIN,BUSINESS; PRI=DS; CPU=5000; INPRI=8; OUTPRI=8; JIN=10; JLIST=17
14:06:59:1  #J700     OFF                          CPU-SEC=25; ELAPSED-MIN=4; MAXPRI=0; NUM-CREATIONS=7
14:08:14:2  #J702     ON     FDDEV,MANAGER,FINDEV,PUB; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
14:09:11:1  #J701     OFF                          CPU-SEC=19; ELAPSED-MIN=4; MAXPRI=0; NUM-CREATIONS=5
14:09:21:4  #J702     OFF                          CPU-SEC=5; ELAPSED-MIN=2; MAXPRI=0; NUM-CREATIONS=1
14:09:41:6  #J703     ON     FDDEV,KELSEY,WCCS,K1; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
14:09:42:2  #J704     ON     FL1JOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=6; OUTPRI=0; JIN=10; JLIST=17
14:10:15:5  #S2896    OFF                          CPU-SEC=48; ELAPSED-MIN=71; MAXPRI=0; NUM-CREATIONS=23
14:10:31:3  #S3147    ON     HANFORET,STU82,HANFORET; PRI=CS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=44; JLIST=44
14:11:33:5  #S3151    ON     OHPROD,MANAGER,ADMIN,COMMON; PRI=CS; CPU=UNLIM;HIPRI; OUTPRI=0; JIN=45; JLIST=45
14:12:30:6  #J704     OFF                          CPU-SEC=9; ELAPSED-MIN=3; MAXPRI=0; NUM-CREATIONS=4
14:13:14:2  #J703     OFF                          CPU-SEC=15; ELAPSED-MIN=4; MAXPRI=0; NUM-CREATIONS=3
14:13:33:4  #J705     ON     FDDEV,MANAGER,FINDEV,PUB; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
14:14:24:3  #J706     ON     ARPROD,KELSEY,WCCS,K1; PRI=DS; CPU=UNLIM; INPRI=8; OUTPRI=0; JIN=10; JLIST=23
14:15:27:0  #S2956    OFF                          CPU-SEC=82; ELAPSED-MIN=56; MAXPRI=0; NUM-CREATIONS=10
14:15:58:3  #J705     OFF                          CPU-SEC=10; ELAPSED-MIN=3; MAXPRI=0; NUM-CREATIONS=1
14:16:13:4  #J707     ON     FL2JOB,BATCH,ADMIN,BUSINESS; PRI=DS; CPU=UNLIM; INPRI=6; OUTPRI=0; JIN=10; JLIST=17
```

```
   TIME      TYPE  JOB#        C00.00              DATE: FRI, DEC,  4,1981     LOGFILE: 2345

                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :51:2  FILE  J 684    $STDLIST.BATCH    .ADMIN      0      1     1280         32 /0      26           0
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                        PROG SEG * SL SEG * MAX STACK * MAX DS * VIRT ST *
 12:5 :51:4  PROC  J 684     0          0         2772        14        147
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :51:6  FILE  J 684    QUISPUDC.PUB      .IRIS       0      1     2            0  /11     3            1
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :51:7  FILE  J 684    SYSUDC  .PUB      .SYS        0      1     35           0  /13     93           6
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :52:1  FILE  J 684    AD310S13.BATCH    .ADMIN      4      2     1604         0  /11     0            0
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                        MAX PRI * CREAT * CPU TIME(S) * ELAPSED (M) *
 12:5 :52:5   OFF  J 684     0        2       175            9
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :53:5  FILE  J 684    $STDLIST.BATCH    .ADMIN      0      0     36           32 /0      33           2
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :53:7  FILE  J 684    $STDIN  .BEACON   .IRIS       4      1     8            24 /0      14           1
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :55:9  FILE  S 2879   $NEWPASS.WEISSHC  .STU85B     0      0     201          0  /13     21           21
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:2  FILE  S 2879   FTNIC   .WEISSHC  .STU85B     4      0     256          0  /13     12           12
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:3  FILE  S 2879   FTNUT1  .WEISSHC  .STU85B     4      0     9            0  /13     6            6
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:4  FILE  S 2879   PROJ1   .WEISSHC  .STU85B     0      1     33           0  /13     95           32
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:4  FILE  S 2879   FTNLIST .         .           1      0     0            16 /61     201          201
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                        PROG SEG * SL SEG * MAX STACK * MAX DS * VIRT ST *
 12:5 :56:4  PROC  S 2879    22         5         6832         8        276
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:5  FILE  S 2879   $STDIN  .         .           0      0     0            16 /61     201          201
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :56:5  FILE  S 2879   $STDLIST.         .           0      0     0            16 /61     201          201
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :57:5  FILE  S 2879   SEGPROC .PUB      .SYS        0      1     144          0  /1      9            9
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                FILE NAME             * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
 12:5 :59:5  FILE  S 2879   $NEWPASS.WEISSHC  .STU85B     0      0     15           0  /2      28           28
                       * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

FROM: FRI, DEC 4, 1981 12:05 PM TO FRI, DEC 4, 1981 2:57 PM

LDEV	JOB COUNT	JOB CPU SEC./CONNECT MIN.			JOB CPU SECONDS			JOB CONNECT MINUTES		
	TOTAL	TOTAL	JOB AVE	STD.DEV.	TOTAL	AVERAGE	STD.DEV.	TOTAL	AVERAGE	STD.DEV.
20	0									
21	0									
22	0									
23	0									
24	0									
25	0									
26	1	.83	.83	.00	5	5.00	.00	6	6.00	.00
27	8	1.61	1.54	1.06	106	13.25	12.82	66	8.25	6.78
28	0									
29	0									
30	0									
31	0									
32	0									
33	0									
34	0									
35	0									
36	0									
37	0									
38	0									
39	0									
40	0									
41	1	1.19	1.19	.00	121	121.00	.00	102	102.00	.00
42	0									
43	1	.75	.75	.00	60	60.00	.00	80	80.00	.00
44	5	.60	.96	.62	47	9.40	4.88	78	15.60	15.83

SLISTLOG A.00.00 DATE: FRI, DEC 4, 1981, 12:06 PM LOGFILE: 2345

TIME	JOB#	DFID	FILENAME	JSNAME, USER . ACCOUNT	ORIG-J#	NUM-I/O	#SECT	COP	PRI	SP#	D-T	C-C	DIS
12:06:19:7	#S2879	#06296	FTN82	WEISSHC,STU85B		51	40	0	8	18	32	OK	0
12:07:46:1	#S2846	#06249	QUADLIST	WINTERJ,FAC		5445	884	0	8	6	32	OK	0
12:08:05:5	#S2864	#06250	FTN50	RIORUD,WHITMAN		278	136	0	8	6	32	OK	0
12:09:07:9	#J677	#06251	$STDLIST	CR212JOB,BATCH,ADMIN		39	32	0	8	6	32	OK	0
12:08:11:6	#S2866	#06259	Y1	OHPROD,MANAGER,STU85B		45	44	0	8	6	32	OK	0
12:08:16:9	#S2867	#06261	Y1	OHPROD,MANAGER,STU85		87	56	0	8	6	32	OK	0
12:08:21:8	#S2868	#06263	Y1	OHPROD,MANAGER,STU84		78	52	0	8	6	32	OK	0
12:08:25:0	#S2869	#06265	Y1	OHPROD,MANAGER,STU84B		45	44	0	8	6	32	OK	0
12:08:30:2	#S2870	#06267	Y1	OHPROD,MANAGER,STU83		84	56	0	8	6	32	OK	0
12:08:31:5	#J680	#06268	$STDLIST	FBTRL,RICHMOSL,STAFF		15	32	0	8	6	32	OK	0
12:08:36:0	#S2871	#06271	Y1	OHPROD,MANAGER,STU82		70	52	0	8	6	32	OK	0
12:08:44:0	#J670	#06214	$STDLIST	FDDEV,MANAGER,FINDEV		136	48	0	8	6	32	OK	0
12:08:45:4	#S2873	#06276	Y1	OHPROD,MANAGER,STU82B		11	32	0	8	6	32	OK	0
12:08:47:7	#J681	#06270	$STDLIST	CR210JOB,BATCH,ADMIN		39	32	0	8	6	32	OK	0
12:08:55:4	#J684	#06291	F422WIDE	AD310JOB,BATCH,ADMIN		123	68	0	8	6	32	OK	0
12:08:57:6	#J684	#06290	$STDLIST	AD310JOB,BATCH,ADMIN		35	36	0	8	6	32	OK	0
12:12:04:2	#J683	#06289	$STDLIST	FDDEV,KELSEY,WCCS		3190	916	0	8	6	32	OK	0
12:17:10:7	#S2862	#06299	EDTLIST	PETSOLL,STU84		182	60	0	8	18	32	OK	0
12:24:26:2	#S2882	#06303	FTN82	WEISSHC,STU85B		51	40	0	8	18	32	OK	0
12:26:42:3	#S2882	#06304	EDTLIST	WEISSHC,STU85B		133	52	0	8	18	32	OK	0
12:42:24:1	#S2846	#06306	LP	WINTERJ,FAC		134	40	0	8	18	32	OK	0
12:48:51:7	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		14	456	7	12	6	32	OK	1
12:49:59:0	#S2862	#06310		PETSOLL,STU84		7	32	0	8	18	32	OK	0
12:50:16:9	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	7	12	6	32	OK	0
12:51:31:1	#S2862	#06312	EDTLIST	PETSOLL,STU84		195	64	0	8	18	32	OK	0
12:51:35:5	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	6	12	6	32	OK	0
12:52:51:6	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	5	12	6	32	OK	0
12:53:45:8	#S2885	#06313	FTN99	HANFORET,STU82		37	36	0	8	18	32	OK	0
12:54:08:0	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	4	12	6	32	OK	0
12:55:23:7	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	3	12	6	32	OK	0
12:56:39:3	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	2	12	6	32	OK	0
12:57:57:5	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	1	12	6	32	OK	0
12:59:17:0	#J677	#06252	PRINTER	CR212JOB,BATCH,ADMIN		1299	456	0	12	6	32	OK	0
13:00:13:7	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	7	11	6	32	OK	0
13:01:08:8	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	6	11	6	32	OK	0
13:02:07:3	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	5	11	6	32	OK	0
13:03:05:6	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	4	11	6	32	OK	0
13:04:01:3	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	3	11	6	32	OK	0
13:04:57:8	#J681	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	2	11	6	32	OK	0
13:05:35:1	#S2903	#06329	EDTLIST	HOWELLAC,STU84B		57	36	0	8	18	32	OK	0
13:05:48:7	#S2888	#06330	FTN82	WEISSHC,STU85B		51	40	0	8	18	32	OK	0
13:05:54:4	#J691	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	1	11	6	32	OK	0
13:06:50:8	#J691	#06277	PRINTER	CR210JOB,BATCH,ADMIN		764	384	0	11	6	32	OK	0
13:09:40:2	#J685	#I3720	$STDIN	UFBTCHCR,BUSENTRY,ADMIN	#S2798	9	8	0	0	11	0	OK	0
13:10:25:5	#J685	#06334	LP	UFBTCHCR,BUSENTRY,ADMIN		119	96	0	8	19	32	OK	0
13:10:31:0	#J685	#06333	$STDLIST	UFBTCHCR,BUSENTRY,ADMIN		36	36	0	8	19	32	OK	0
13:12:33:1	#J686	#I3721	$STDIN	FBTRL,RICHMOSL,STAFF	#S2904	25	12	0	0	11	0	OK	0
13:12:42:0	#J687	#I3722	$STDIN	FL9JOB,BATCH,ADMIN	#S2798	8	8	0	0	2	0	OK	0
13:13:02:3	#J686	#06335	$STDLIST	FBTRL,RICHMOSL,STAFF		15	32	0	8	6	32	OK	0
13:14:59:9	#S2881	#06300	STENOPLT	FILLADF,STAFF		166	68	0	8	24	32	OK	0
13:15:06:4	#J687	#06337	PRINT	FL9JOB,BATCH,ADMIN		260	176	0	8	19	32	OK	0
13:15:11:8	#J687	#06336	$STDLIST	FL9JOB,BATCH,ADMIN		35	32	0	8	19	32	OK	0
13:19:39:5	#S2888	#06340	FTN82	WEISSHC,STU85B		51	40	0	8	18	32	OK	0
13:19:48:4	#J688	#I3759	$STDIN	FBTRL,RICHMOSL,STAFF	#S2904	26	8	0	0	13	0	OK	0

DEPARTMENT OF REGIONAL ECONOMIC EXPANSION

OTTAWA

WED, JAN 6, 1982, 10:22 PM

LOGFILE SUMMARY FOR

LOG2345.PUB

TYPE NO.	TYPE	# RECORDS
0	LOG FAILURE	0
1	SYSTEM STARTUP	0
2	JOB INITIATION	86
3	JOB TERMINATION	76
4	PROCESS TERMINATION	735
5	FILE CLOSE	7732
6	SYSTEM SHUTDOWN	0
7	POWER FAILURE	0
8	SPOOLING LOG RECORD	232
9	LINE DISCONNECTION	0
10	LINE CLOSE	0
11	I/O ERROR	0
12	DISC PHYS MNT/DSMNT	0
13	DISC LOGICAL MNT/DSMT	0
14	LABELLED TAPE	0
		8861

START TIME: FRI, DEC 4, 1981, 12:05 PM

STOP TIME: FRI, DEC 4, 1981, 2:57 PM

SYSTEM RECORDS: 650 USER RECORDS: 8211

Whitman College Computer Services

Job Accounting System

Summary for: LOG2345.PUB

Start Time: FRI, DEC 4, 1981, 12:05 PM

Stop Time: FRI, DEC 4, 1981, 2:57 PM

TYPE NO.	TYPE	# RECORDS
0	LOG FAILURE	0
1	SYSTEM STARTUP	0
2	JOB INITIATION	86
3	JOB TERMINATION	76
4	PROCESS TERMINATION	735
5	FILE CLOSE	7732
6	SYSTEM SHUTDOWN	0
7	POWER FAILURE	0
8	SPOOLING LOG RECORD	232
9	LINE DISCONNECTION	0
10	LINE CLOSE	0
11	I/O ERROR	0
12	DISC PHYS MNT/DSMNT	0
13	DISC LOGICAL MNT/DSMT	0
14	LABELLED TAPE	0
		8861

SYSTEM Records: 650
USER Records: 8211
8861

Date Processed: WED, DEC 30, 1981, 5:26 PM

LOG2345,PUB

PAGE 1 DEPARTMENT OF REGIONAL ECONOMIC EXPANSION WED, JAN 6, 1982, 11:47 PM

SUMMARY REPORT FOR ADMIN FOR OCTOBER 1981

COST CENTRE:H42 GROUP:BUSINESS

JOB	DATE	TIME	JOBNAME	USER		CPU/S	CON/M	PROC	CD/SEG	SWAP			PROC-I/O COST	JOB COST
J685	4 /12/81	13:8	UFBTCHCR	BUSENTRY	D	12	2	4	31	810			$0.75	
				DISC:		26 FILES		1253 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		155 LINES		CARD READER:	0 FILES	0 RECORDS	$1.00	$1.75
J687	4 /12/81	13:12	FL9JOB	BATCH	D	8	2	5	49	492			$0.81	
				DISC:		17 FILES		320 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		295 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.81
J689	4 /12/81	13:29	UFBATCHO	BUSENTRY	D	8	2	4	23	691			$0.61	
				DISC:		24 FILES		489 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		132 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.61
J691	4 /12/81	13:40	UFUPDTCA	BUSENTRY	D	7	2	4	24	790			$0.62	
				DISC:		28 FILES		263 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		54 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.62
J692	4 /12/81	13:44	UFUPDTCA	BUSENTRY	D	8	2	4	24	790			$0.63	
				DISC:		28 FILES		250 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		64 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.63
J694	4 /12/81	13:45	GL1AJOB	BATCH	D	15	2	8	92	1382			$1.51	
				DISC:		41 FILES		396 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		3 FILES		227 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$1.51
J695	4 /12/81	13:49	GL1AJOB	BATCH	D	14	4	8	92	1136			$1.50	
				DISC:		41 FILES		524 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		3 FILES		241 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$1.50
J698	4 /12/81	13:57	GL1AJOB	BATCH	D	13	5	8	92	1136			$1.50	
				DISC:		41 FILES		278 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		3 FILES		162 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$1.50
J699	4 /12/81	14:1	UFUPDTCA	BUSENTRY	D	7	2	4	24	790			$0.62	
				DISC:		28 FILES		218 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		2 FILES		50 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.62
J700	4 /12/81	14:3	GL1AJOB	BATCH	D	25	4	8	92	1136			$1.66	
				DISC:		41 FILES		969 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		3 FILES		549 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$1.66
J701	4 /12/81	14:5	UFDSJENT	BUSENTRY	D	19	4	6	44	1093			$1.13	
				DISC:		36 FILES		2924 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		3 FILES		376 LINES		CARD READER:	0 FILES	0 RECORDS	$2.00	$3.13
J704	4 /12/81	14:9	FL1JOB	BATCH	D	9	3	5	53	465			$0.87	
				DISC:		12 FILES		313 BLOCKS		TERMINAL:	0 RECORDS			
				PRINTER:		1 FILES		34 LINES		CARD READER:	0 FILES	0 RECORDS	$0.00	$0.87

```
PAGE   3                                   DEPARTMENT OF REGIONAL ECONOMIC EXPANSION                    WED, JAN  6, 1982, 11:47 PM
                                                                --------
              ---           ---------      SUMMARY REPORT FOR ADMIN     FOR OCTOBER     1981
 COST CENTRE:H42        GROUP:BUSINESS                         --------
              ---           ---------
 JOB   DATE     TIME JOBNAME  USER      G CPU/S  CON/M  PROC  CD/SEG     SWAP                                      PROC-I/O COST   JOB COST

        BUSINESS  GROUP TOTALS         361 CPU/SEC       78 CON/MIN    151 PROCESSES
                                      1540 CODE/SEG   22718 SWAP        24 JOBS      0 SESSIONS
                                DISC:   786 FILES   19883 BLOCKS       TERMINAL:          0 RECORDS
                             PRINTER:    61 FILES    5980 LINES     CARD READER:          0 FILES          0 RECORDS
                                             PROCESSING COSTS:    $28.48  I/O COSTS:        $9.00

              H42 COST CENTRE TOTALS   361 CPU/SEC       78 CON/MIN    151 PROCESSES
                                      1540 CODE/SEG   22718 SWAP        24 JOBS      0 SESSIONS
                                DISC:   786 FILES   19883 BLOCKS       TERMINAL:          0 RECORDS
                             PRINTER:    61 FILES    5980 LINES     CARD READER:          0 FILES          0 RECORDS
                                             PROCESSING COSTS:    $28.48  I/O COSTS:        $9.00

        ADMIN     ACCOUNT TOTALS       361 CPU/SEC       78 CON/MIN    151 PROCESSES
                                      1540 CODE/SEG   22718 SWAP        24 JOBS      0 SESSIONS
                                DISC:   786 FILES   19883 BLOCKS       TERMINAL:          0 RECORDS
                             PRINTER:    61 FILES    5980 LINES     CARD READER:          0 FILES          0 RECORDS
                                             PROCESSING COSTS:    $28.48  I/O COSTS:        $9.00
```

PAGE 4 DEPARTMENT OF REGIONAL ECONOMIC EXPANSION WED, JAN 6, 1982, 11:47 PM

COST CENTRE: --- GROUP: --------

SUMMARY REPORT FOR ADMIN FOR OCTOBER 1981

JOB	DATE	TIME	JOBNAME	USER	Q	CPU/S	CON/M	PROC	CD/SEG	SWAP	PROC-I/O COST	JOB COST

	NAME	JOBS (COUNT)	JOBS COST	SESSIONS (COUNT)	SESSIONS COST
USER TOTALS:	BUSENTRY	8	$16.17	0	$0.00
	BATCH	16	$21.31	0	$0.00

	NAME	JOBS (COUNT)	JOBS COST	SESSIONS (COUNT)	SESSIONS COST
JOB TOTALS:	UFBTCHCR	1	$1.75	0	$0.00
	FL9JOB	2	$1.57	0	$0.00
	UFBATCHO	1	$0.61	0	$0.00
	UFUPDTCA	3	$1.87	0	$0.00
	GL1AJOB	11	$17.07	0	$0.00
	UFDSJENT	1	$3.13	0	$0.00
	FL1JOB	1	$0.87	0	$0.00
	FL2JOB	2	$1.80	0	$0.00
	UFDISTAR	1	$1.49	0	$0.00
	UFUPDTPY	1	$7.32	0	$0.00

H-P PROCESSING INVOICE FOR COST CENTRE H42, OCT 1981
WED, JAN 6, 1982, 11:43 PM

ACCOUNT	GROUP	COST
	PUB	$1.53
AAA		$1.53
	BUSINESS	$37.48
ADMIN		$37.48
	COMMON	$5.38
	PUB	$1.81
FINDEV		$7.19
	FILLADF	$4.70
	RICHMOSL	$4.70
STAFF		$9.40
	GILLILLK	$10.66
	HANFORET	$6.69
	RITCHEKO	$5.21
STU82		$22.56
	HODELDB	$49.00
	ROSIKSL	$2.56
STU83		$51.56
	PENGRADB	$2.03
STU84		$2.03
	COTTREFM	$63.16
	HANSONJA	$1.11
	HOWELLAC	$0.38
	HURSTSA	$41.55

```
TIME  TYPE       JOB#    * * * * * * * * * * * * * * FRI, DEC  4, 1981 * * * * * * * * * * * * * * * * * * * * * * * * * * * LOG2345 * *

12:55 FILE CLOSE #S2883 LDEV=13  MDISC  B1       .COTTREFM,STU84B    #RECORDS=69          #BLOCKS=23
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=24
12:55 FILE CLOSE #S2883 LDEV=5   MDISC  K3381255.COTTREFM,STU84B    #RECORDS=90          #BLOCKS=19
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=224
12:55 SPOOL FILE #J677  #06252 PRINTER  CR212JOB,BATCH    ,ADMIN    COPIES=3   ,PRI=12,RECS=1299      ,SECTORS=456
12:55 FILE CLOSE  SYS   LDEV=3   MDISC          ,BATCH    ,ADMIN    #RECORDS=1297        #BLOCKS=107
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=456
12:55 FILE CLOSE #J553  LDEV=1   MDISC  COMPDATA.PUB      ,SYS      #RECORDS=2           #BLOCKS=6
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=25
12:55 FILE CLOSE #J553  LDEV=1   MDISC  COMPDATA.PUB      ,SYS      #RECORDS=2           #BLOCKS=6
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=25
12:56 PROCESS    #S2885   0 PROG-SEGMENTS,   0 SL-SEGMENTS,   166 VIRTUAL-MEM-SECTORS, MAX-STACK(WRDS)=2772 ,MAX-XDS(SECTORS)=14
12:56 FILE CLOSE #S2885 LDEV=5   MDISC  UDCSTU   .PUB      ,IRIS     #RECORDS=72          #BLOCKS=11
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=24
12:56 FILE CLOSE #S2885 LDEV=13  MDISC  SYSUDC   .PUB      ,SYS      #RECORDS=93          #BLOCKS=6
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=35
12:56 FILE CLOSE #S2885 LDEV=11  MDISC  $OLDPASS,HANFORET,STU82     #RECORDS=0           #BLOCKS=0
                                       DOMAIN:  NEW          DISPOSITION:  DELETE        #SECTORS=10
12:56 JOB TERM   #S2885    12 PROCESSES RUN,          25 CPU SEC USED,          14 ELAPSED MINUTES
12:56 FILE CLOSE #S2885 LDEV=47  TERM   $STDLIST,        ,                   #RECORDS=816         #BLOCKS=816
12:56 FILE CLOSE #S2885 LDEV=17  TERM   $STDIN   .       .                   #RECORDS=816         #BLOCKS=816
**JOB SUMMARY ** #S2885 LDEV=47  HANFORET,STU82   12:42-12:56 PROG-RUNS=13   STACK:      8240=MAX,      5360=AVG; CPU=25        0=CS
                        TRANSFERRED       27 RECORDS TO DEVICE 1   MOVING HEAD DISC
                        TRANSFERRED      635 RECORDS TO DEVICE 2   MOVING HEAD DISC
                        TRANSFERRED      447 RECORDS TO DEVICE 3   MOVING HEAD DISC
                        TRANSFERRED      409 RECORDS TO DEVICE 5   MOVING HEAD DISC
                        TRANSFERRED      469 RECORDS TO DEVICE 11  MOVING HEAD DISC
                        TRANSFERRED      591 RECORDS TO DEVICE 13  MOVING HEAD DISC
                        TRANSFERRED       37 RECORDS TO DEVICE 18  LINE PRINTER
                        TRANSFERRED    21049 RECORDS TO DEVICE 47  TERMINAL
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  B1       .COTTREFM,STU84B    #RECORDS=0           #BLOCKS=0
                                       DOMAIN:  NEW          DISPOSITION:  NO CHANGE     #SECTORS=5
12:56 FILE CLOSE #S2883 LDEV=13  MDISC  B1       .COTTREFM,STU84B    #RECORDS=0           #BLOCKS=0
                                       DOMAIN:  OLD          DISPOSITION:  DELETE        #SECTORS=24
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  K3381255.COTTREFM,STU84B    #RECORDS=89          #BLOCKS=16
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=224
12:56 FILE CLOSE #S2883 LDEV=11  MDISC  B1       .COTTREFM,STU84B    #RECORDS=69          #BLOCKS=23
                                       DOMAIN:  NEW          DISPOSITION:  SAVE          #SECTORS=24
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  K3381255.COTTREFM,STU84B    #RECORDS=108         #BLOCKS=12
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=224
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  K3381255.COTTREFM,STU84B    #RECORDS=3           #BLOCKS=3
                                       DOMAIN:  OLD          DISPOSITION:  DELETE        #SECTORS=224
12:56 PROCESS    #S2883  12 PROG-SEGMENTS,   0 SL-SEGMENTS,   133 VIRTUAL-MEM-SECTORS, MAX-STACK(WRDS)=8240 ,MAX-XDS(SECTORS)=14
12:56 FILE CLOSE #S2883 LDEV=61  TERM   $STDIN   .       .                   #RECORDS=2054        #BLOCKS=2054
12:56 FILE CLOSE #S2883 LDEV=61  TERM   $STDLIST.        .                   #RECORDS=2054        #BLOCKS=2054
12:56 FILE CLOSE #S2883 LDEV=61  TERM   EDITOUT  .       .                   #RECORDS=2054        #BLOCKS=2054
12:56 FILE CLOSE #S2883 LDEV=61  TERM   EDITIN   .       .                   #RECORDS=2054        #BLOCKS=2054
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  FORTRAN  .PUB     ,SYS      #RECORDS=15          #BLOCKS=15
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=384
12:56 FILE CLOSE #S2883 LDEV=5   MDISC  FTNUSL   .COTTREFM,STU84B    #RECORDS=0           #BLOCKS=0
                                       DOMAIN:  NEW          DISPOSITION:  NO CHANGE     #SECTORS=11
12:56 SPOOL FILE #J677  #06252 PRINTER  CR212JOB,BATCH    ,ADMIN    COPIES=2   ,PRI=12,RECS=1299      ,SECTORS=456
12:56 FILE CLOSE  SYS   LDEV=3   MDISC          ,BATCH    ,ADMIN    #RECORDS=1297        #BLOCKS=107
                                       DOMAIN:  OLD          DISPOSITION:  NO CHANGE     #SECTORS=456
12:56 FILE CLOSE #S2883 LDEV=11  MDISC  $NEWPASS,COTTREFM,STU84B    #RECORDS=17          #BLOCKS=17
                                       DOMAIN:  NEW          DISPOSITION:  NO CHANGE     #SECTORS=201
```

```
TIME  TYPE       JOB:   * * * * * * * * * * * * * FRI, DEC  4, 1981 * * * * * * * * * * * * * * * * * * * * * * * * * LOG2345 * *

12:57 FILE CLOSE #S2890 LDEV=11  MDISC  GUARD11 ,GUARD    ,IRIS      #RECORDS=1           #BLOCKS=1
                                       DOMAIN:  OLD         DISPOSITION:  NO CHANGE    #SECTORS=468
12:57 FILE CLOSE #S2890 LDEV=13  MDISC  GUARD12 ,GUARD    ,IRIS      #RECORDS=1           #BLOCKS=1
                                       DOMAIN:  OLD         DISPOSITION:  NO CHANGE    #SECTORS=30
12:57 FILE CLOSE #S2890 LDEV=13  MDISC  GUARD13 ,GUARD    ,IRIS      #RECORDS=1           #BLOCKS=1
                                       DOMAIN:  OLD         DISPOSITION:  NO CHANGE    #SECTORS=64

      0 LOG FILE ERRORS
     10 JOB INITIATIONS
     12 JOB TERMINATIONS
    111 PROCESS COMPLETIONS
   1295 FILE CLOSES
     31 SPOOFLE FINISHED
```

12/ 4/81 THRU 12/ 4/81 ANALYZED ON WED, JAN 6, 1982, 12:08 AM

USER.ACCOUNT	# JOBS	# SESN	CONNECT HOURS	CPU TIME (HOURS) BS	CS	DS	ES	PRIME	TOTAL	STACK MAX	STACK AVE	# PROC	DISC RECORDS	PRINTER LINES	MAG TAPE RECORDS	READR CARDS	PUNCH CARDS
MANAGER .AAA	0	1	.20		.01			.01	.01	5871	4321	2	126.	0	0	0	0
WINTERJ .FAC	0	1	2.32					.00	.11	13237	5075	4	5098.	5579	0	0	0
HAMFORET.STUR2	0	1	.23		.01			.01	.01	8240	5360	13	2578.	37	0	0	0
WEISSHC .STU85B	0	2	.33		.01			.01	.01	8240	5591	8	11327.	235	0	0	0
NYHAGEN .CCS	0	1	.10		.00			.00	.00	5558	4165	2	329.	0	0	0	0
@ .@	0	6	3.18		.02			.02	.13	13237	5229	29	19458.	5851	0	0	0

! SUMMARY OF I/O ERRORS !

BETWEEN FRI, DEC 4, 1981, 12:05 PM - FRI, DEC 4, 1981, 2:57 PM
THERE WERE 0 POWER FAILURES
THE SYSTEM WAS SHUT DOWN 0 TIMES
THE SYSTEM WAS RESTARTED 0

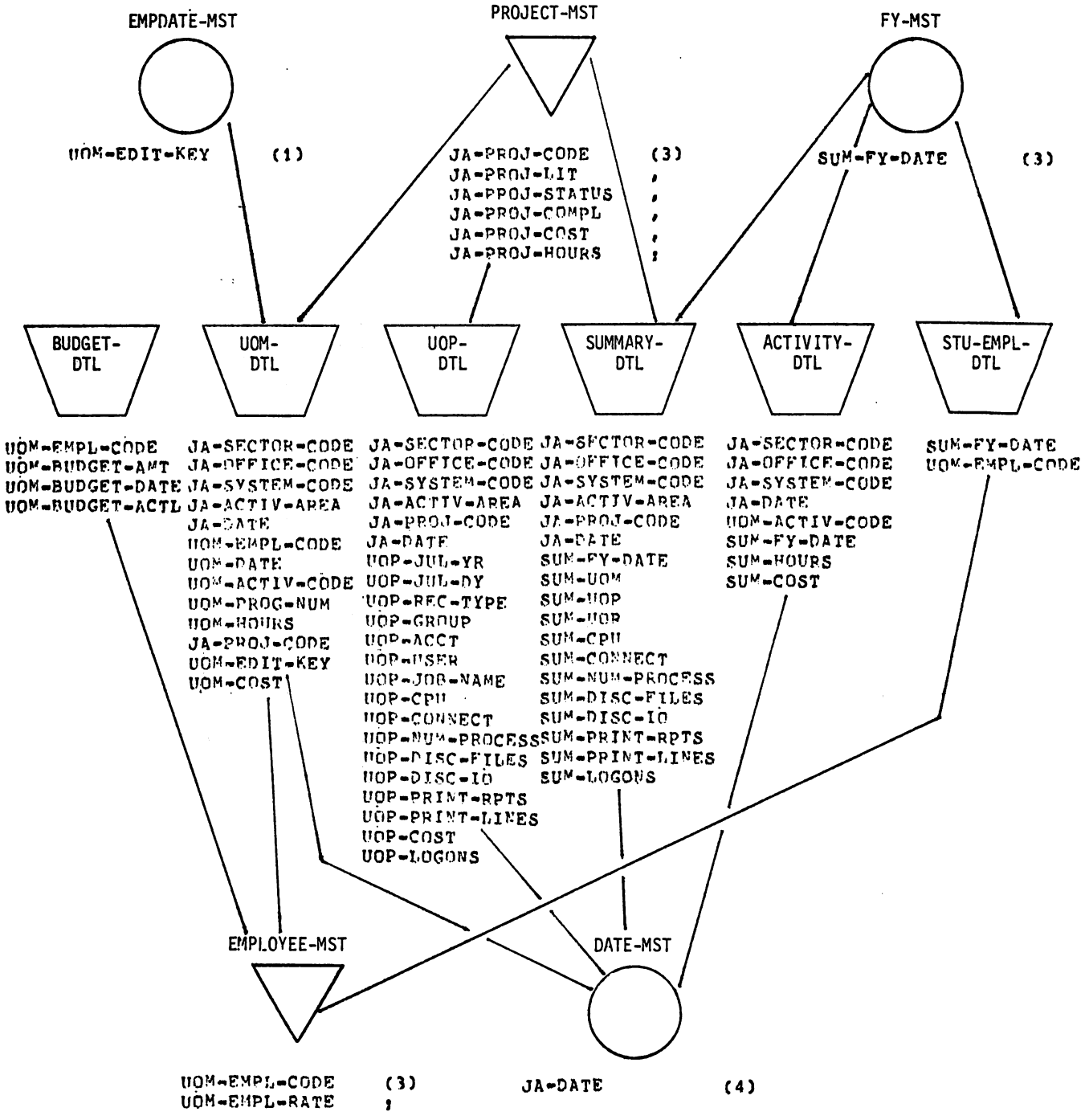
REPORT FROM SUMMARY FILE FILESUM.LOGFILES.SYS FRI, DEC 4, 1981, 12:05 PM TO FRI, DEC 4, 1981, 12:57 PM
WED, JAN 6, 1982, 12:06 AM SHORT FILE ACTIVITY REPORT BY FILE NAME PAGE 1

FILE/USER NAME	DISPOSITION	LDN	SECTORS	CLOSES:TTL	JOBS	SESS	REC-USE:TTL	JOBS	SESS	BLKS:-TTL	JOBS	SESS
CATALOG.PUB.SYS	PERM,SAME	1	910	4		4	576		576	36		36
COMMAND.PUB.SYS	PERM,SAME	3	168	9		9	41		41	34		34
CONFDATA.PUB.SYS	PERM,SAME	1	25	52	52		104	104		312	110	
EDITOR.PUB.SYS	PERM,SAME	2	289	29		29	261		261	261		261
FORTRAN.PUB.SYS	PERM,SAME	5	384	15		15	225		225	225		225
LOG2344.PUB.SYS	PERM,SAME	11	2046	1			8326			1022		
LOG2345.PUB.SYS	NEW,PERM	11	128	1			0			0		
QUERY.PUB.SYS	PERM,SAME	13	371	2		1	156		34	156		34
SEGPROC.PUB.SYS	PERM,SAME	1	144	17		17	153		153	153		153
SPL.PUB.SYS	PERM,SAME	2	372	19		10	1965		300	1965		300
SPLINTR.PUB.SYS	PERM,SAME	3	133	10		10	78		78	74		74
SYSUDC.PUB.SYS	PERM,SAME	13	35	12	2	10	1123	193	930	73	200	60

APPENDIX B

Table of Contents

The DIARY Data Base



Appendix B - I.1

91	M.A.R.C.U.S. Utilities System
AD	Student Admissions System
AL	Alumni Records System
AP	Accounts Payable System
AR	Accounts Receivable
AS	Academic Support System
AT	Art
AY	Astronomy
BD	Budget Analysis System
BG	Beacon/Guardian System
BI	Biology
BP	Bibliographic Retrieval System
CG	Class Grading System
CH	Chemistry
CL	Contact Log System
CO	General Purpose Consulting System
CR	Class Registration System
CS	Central Supply Inventory System
DR	Drama
EC	Economics
ED	Education
EN	Environmental Studies
ER	Employee Records System
FA	Student Financial Aid System
FD	Financial Development System
FL	FISL Maintenance System
FS	Freshman Seminar
GE	Geology
GL	General Ledger System
GR	Gift Records System 07/02/81-06/30/82
HI	History
IB	Interdepartmental Billing System
JA	TROIKA -- HP3000 Computer Job Accounting
LA	Languages
LC	Library Circulation System
MA	Mathematics
ML	Central Mailing System
MU	Music
NL	NDL Maintenance System
OH	Overhead
PC	Psychology
PE	Physical Education
PH	Philosophy
PS	Political Science
PT	General Plotter/Graphics System
PY	Payroll System
PZ	Physics
RE	Religion
SA	Sociology/Anthropology
SC	Treasurer's Securities System.
SH	Student Housing System

Whitman College Computer Center

Staff Activity Report

Date	09/07/81
ID	G4

Name:

Activity	# Hours	Project	Comments
JAI20	3.5		Testing
OH-M	1.0		Staff meeting
CG-MA	3.5	13281	Analysis w/ Barb Botz, Wayne
AD-PS	.5		Fire fighting - Admissions
Total Hours	8.5		

Appendix B - II.2

Type of Work	Staff Activity Report		Corr. Job/Session Logon
PRODUCTION	SS-PS*	SS-G	PYDDDnSS (after sys install)
	SS-K	SS-TS	SSPROD (before or after inst)
	OH-A	OH-AM	SSnnnJOB (after sys install)
	OH-SM	OH-C	
	OH-T	OH-S	
	OH-VS	OH-M	
DEVELOPMENT	SS-P**	SS-SA**	DYDDDnSS (after sys. install)
	SSnnn**	SS-D**	SSDEV (before installed)
MAINTENANCE	SS-MA	SS-D***	MYDDDnSS (after sys install)
	SS-P***	SSnnn***	SSLOOK (after sys install)

Underlining indicates literal value

* - SR number may accompany activity code

** - abscence of SR number implies development on uninstalled sys

***-- SR number must accompany activity code

General Format of Service Request (SR) numbers : XYDDDNSS, where

X= "D" (development)
"M" (maintenance) or
"P" (production)

Y= "0" through "9", the current year of the decade in which SR was approved

DDD= Julian day of the year Y

N= sequence number of an SR recieved on the Julian day

SS= any valid system code, indicating the system in which work is being performed on behalf of SR

The other logons SSLOOK, SSDEV, and SSPROD are appropriate where SRs are not, as shown above.

LOGON # : #SR79 JAFROD
REPORT : JA323/F906
PRIVACY : CONFIDENTIAL
RPT DATE: 07/08/81

Whitman College Computer Center

Monthly Manpower Summary by Employee

USER: MANAGER
RUN DATE: 07/08/81
RUN TIME: 11:06:39
PAGE: 10

******* January 1981 - June 1981 *******

Amy Galpin

Manhours	%	Code	Activity
89.5	25.2	CL103	Contact Log System Programming
28.0	7.8	CL310	Contact Log System Programming
.5	.1	CL408	Contact Log System Programming
28.0	7.8	JA-AM	Job Accounting System Shop Management/Meeting
1.0	.2	JA-C	Job Accounting System Computer Operation
25.0	7.0	JA-K	Job Accounting System Key Entry/Clerical Support
14.0	3.9	JA-MA	Job Accounting System Maintenance Analysis/Meeting
5.5	1.5	JA***	Job Accounting System Programming
8.0	2.2	JA-SA	Job Accounting System System Design/Analysis/Meetin
2.0	.5	JA-T	Job Accounting System Trainng/Education/Professiona
6.0	1.6	OH-G	Overhead General Activities
64.0	18.0	OH-MA	Overhead Maintenance Analysis/Meeting
21.0	5.9	OH***	Overhead Programming
43.0	12.1	OH-T	Overhead Trainng/Education/Professiona
4.0	1.1	OH-TS	Overhead Technician Support
10.5	2.9	SR170	Student Records System Programming
4.0	1.1	SR175	Student Records System Programming
.5	.1	UT-MA	Computer Center Utilities Maintenance Analysis/Meeting
.5	.1	UT145	Computer Center Utilities Programming

355.0 Total Manhours

LOGON # : #S400/#S31
REPORT : JA405/F602
PRIVACY : CONFIDENTIAL
RPT DATE: 01/03/82

TROIKA -- HP3000 Computer Job Accounting
Whitman College Computer Center

F602 - UOP Monthly Summary by Area

Actual Usage for December , 1981

USER: GALPIN
RUN DATE: 01/03/82
RUN TIME: 19:45:49
PAGE: 1

	# RUNS	CON/M	CPU/S	# RPTS	PRINT LINES	# FILES	DISC I/O
Administrative Production							
Admissions Office	117	1,354	2,251	124	6,958	1,268	62,260
The Registrar	247	5,566	4,377	91	68,800	1,612	203,282
Financial Aid Services	11	31	194	2	2,815	59	11,236
Financial Development	388	2,091	15,381	104	102,456	2,348	878,625
Housing Office	3	40	8	1	20	27	187
Business Office	1,168	3,004	5,269	424	64,670	7,341	378,390
Subtotal	1,934	12,086	27,480	746	245,719	12,655	1,533,980
Academic Support							
Faculty	533	4,410	7,969	123	38,186	4,890	231,282
Students	5,593	21,349	22,100	859	102,119	30,887	568,780
Curriculum/Organizations	218	947	788	15	1,511	1,072	15,979
Subtotal	6,344	26,706	30,857	997	141,816	36,849	816,041
Computer Services							
Software Development	969	6,273	9,780	269	137,450	6,834	257,619
Overhead Support	387	2,965	5,380	142	37,693	2,417	79,890
Subtotal	1,356	9,238	15,160	411	175,143	9,251	337,509
Miscellaneous							
HP Users Group	1	1	1	0	0	1	2
Other	0	0	0	0	0	0	0
Subtotal	1	1	1	0	0	1	2
Total for Month	9,635	48,031	73,498	2,154	562,678	58,756	2,687,532
Batch Jobs	1,563	2,338	26,672	721	330,134	8,577	1,421,646
Interactive Terminal Sessions	8,072	45,693	46,826	1,433	232,544	50,179	1,265,886

LOGON # : #S400/#S31
REPORT : JA405/F602
PRIVACY : CONFIDENTIAL
RPT DATE: 01/03/82

TROIKA -- HP3000 Computer Job Accounting
Whitman College Computer Center

F602 - UOP Monthly Summary by Area

Percentage Usage for December , 1981

USER: GALPIN
RUN DATE: 01/03/82
RUN TIME: 19:45:50
PAGE: 2

	# RUNS	CON/M	CPU/S	# RPTS	PRINT LINES	# FILES	DISC I/O
Administrative Production							
Admissions Office	1.2	2.8	3.0	5.7	1.2	2.1	2.3
The Registrar	2.5	11.5	5.9	4.2	12.2	2.7	7.5
Financial Aid Services	.1	.0	.2	.0	.5	.1	.4
Financial Development	4.0	4.3	20.9	4.8	18.2	3.9	32.6
Housing Office	.0	.0	.0	.0	.0	.0	.0
Business Office	12.1	6.2	7.1	19.6	11.4	12.4	14.0
Subtotal	20.0	25.1	37.3	34.6	43.6	21.5	57.0
Academic Support							
Faculty	5.5	9.1	10.8	5.7	6.7	8.3	8.6
Students	58.0	44.4	30.0	39.8	18.1	52.5	21.1
Curriculum/Organizations	2.2	1.9	1.0	.6	.2	1.8	.5
Subtotal	65.8	55.6	41.9	46.2	25.2	62.7	30.3
Computer Services							
Software Development	10.0	13.0	13.3	12.4	24.4	11.6	9.5
Overhead Support	4.0	6.1	7.3	6.5	6.6	4.1	2.9
Subtotal	14.0	19.2	20.6	19.0	31.1	15.7	12.5
Miscellaneous							
HP Users Group	.0	.0	.0	.0	.0	.0	.0
Other	.0	.0	.0	.0	.0	.0	.0
Subtotal	.0	.0	.0	.0	.0	.0	.0
Total for Month	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Batch Jobs	16.2	4.8	36.2	33.4	58.6	14.5	52.8
Interactive Terminal Sessions	83.7	95.1	63.7	66.5	41.3	85.4	47.1

Online Database: Design and Optimization

Robert B. Garvey
Witan Inc.
Kansas City, Missouri

CONTENTS

FOUNDATION

A system language, GOALS, will be introduced to render systems and components.

A general set of principles will be presented incorporating the components and structures inherent in a structured system. The use of these components in the system life cycle and as a documentation system will evolve.

A general system architecture will be presented and an approach to interactivity will be discussed.

The detailed use of callable programs in the 3000 environment will be discussed with emphasis on improvement of system performance.

I am going to assume that you are first time users of a 3000 that you want to write online database systems, that you do not have some of the more typical real world problems like a conversion from another machine and that you are going to use VPLUS and IMAGE. I don't care what language you use unless it is RPG in which case much of what I say will not be true.

© Copyright Witan Inc. 64113

GOALS: A System Language

GOALS was designed to meet the following criteria:

- Provide good documentation throughout the lifecycle
- Ease maintainence
- Expedite development
- Provide users early understanding of System functions and restraints
- Improve project management and reporting
- Reduce resources required
- Optimize System performance and quality

Many of the above criteria can be achieved through reasonable structuring of the system . However many of the structuring techniques that are now popular are simply more trouble than they are worth. Yourdon, Jackson and certainly IBM's HIPO involve more work involve more work in their maintainence than rewards merit. Warnier comes closest to being worthwhile but cannot be reasonably maintained in machine sensible form.

GOALS will be described as a methodology only because it does what the popular "Methodologies" tout, and much more. We do not feel that any of the methodologies should be considered ends in themselves and more sacred than the system at hand. Once the principles are learned and applied the implications should be obvious and the apparent need for a methodology forgotten.

Documentation

General Statement

The purpose of documentation is to assist in the analysis, design, program design, maintenence and operation of a system. To those ends software documentation must be flexible, easily modifiable, current and easy to read. Witan has developed a system of documentation called GOALS which uses simple text files associated through control numbers to meet the criteria listed above. The following sections describe

the general features of the structural notation used in GOALS and the General system structure used in system projects.

GOALS is used throughout the life of a project. It is used:

1. To state requirements
2. Render flow and components in the analysis phase
3 To develop, test and render a general design
4. As a pseudo code or structured English for detail design
5. As a high level programming language
6. As a project network descriptor.

GOALS: Structural Notation

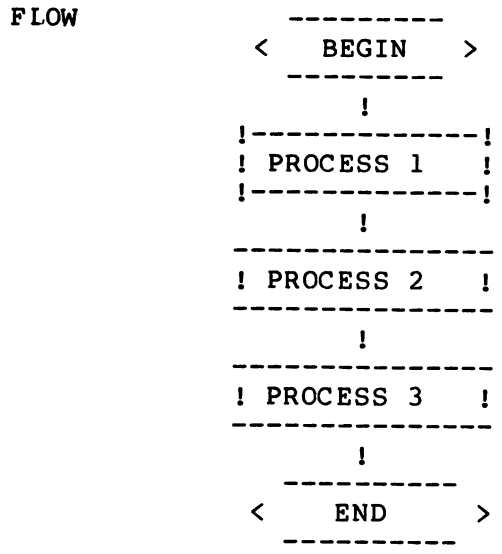
Formal structuring permits three primitive operations: Sequence, Repetition and Alternation. Structural Notation was developed to meet the criteria of formal systems in a generalized way and was guided by the assumption that systems must be rendered in a machine sensible form. GOALS relies upon text sequences and key words as its basis. Structural Notation is the basis of the syntax of GOALS.

Following are the representations of the primitive structures using flowcharts and GOALS. The word PROCESS is used to represent a step, a process or an item depending on the use of the notation at the time.

GOALS Primitive Structures

SEQUENCE

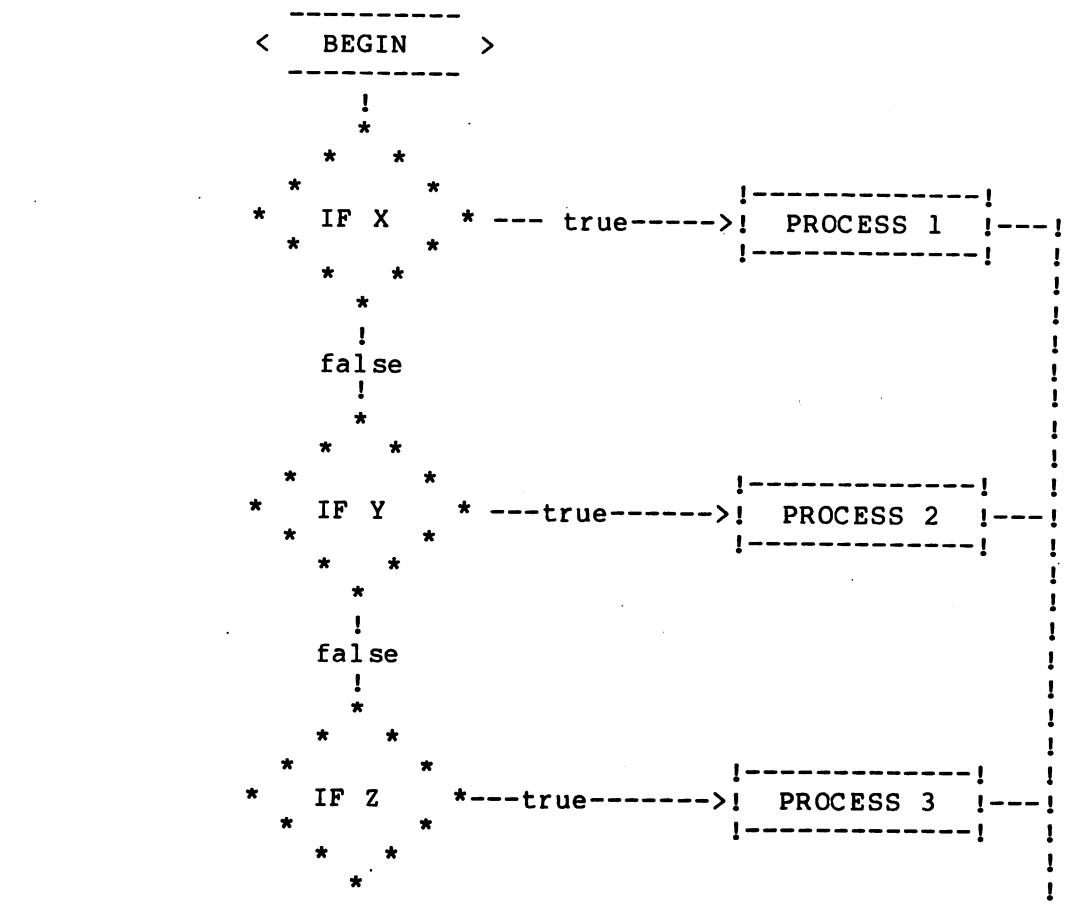
FLOW

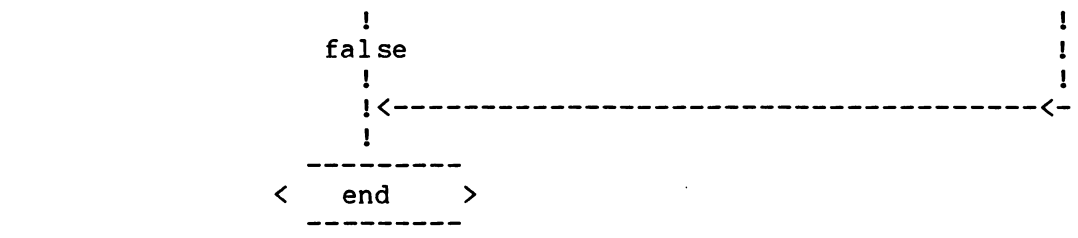



```
GOALS       1  PROCESS 1
            2  PROCESS 2
            3  PROCESS 3
```

ALTERNATION

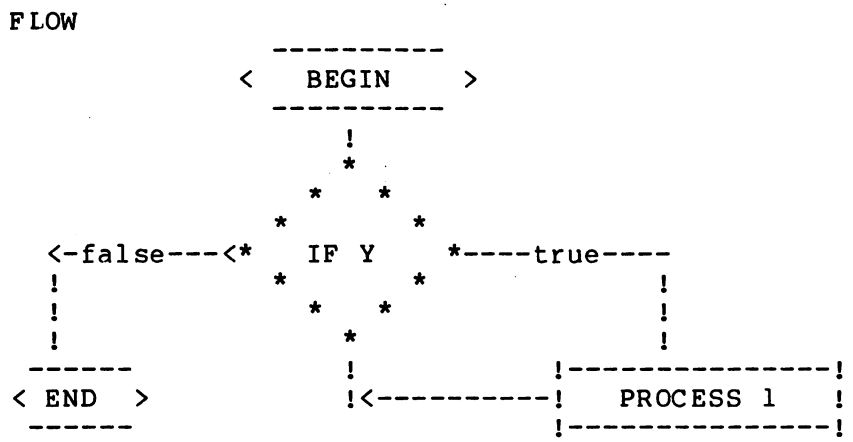
FLOW






```
GOALS       IF X IS TRUE
               PROCESS 1
            IF Y IS TRUE
               PROCESS 2
            IF Z IS TRUE
               PROCESS 3
            .
```

REPETITION




```
GOALS       WHILE  Y <<  IS TRUE  >>
               PROCESS 1
                  PROCESS 1A
                  PROCESS 1B
                  PROCESS 1C
            !
```

The exclamation point is used to signify control in the WHILE loop. If the condition is met the control passes to the statement following the (!) on the same level. If the condition is met the control passes to the first statement following the condition. Processes 1A through 1C were added to show a simple subsequence.

Data Structuring

GOALS is also used to represent data structure. As with control structure there are three general structures which can be represented.

Data items listed line after line represent sequence:

1. item-1
2. item-2
3. item-3

Subsequences are represented as sequences on a level below the item of which they are are a part.

1. item-1
   1A. item-1A
   1B. item-1B
   1C. item-1C
2. item-2
3. item-3

Repetition in data structuring can be represented by "(S)" at the end of the item name which is repeated, this can take the form an expression [i.e., (0>s<15)].

item-1(S)
  item-1

Example: a file of accounts

  Account File
Account(s)
  Account
  Account number

Name
Address(s)
  Address type (h=home, w=work)
  Street number
  Direction
  Street name
  Affix
Amount due
Order(s)
  Order number
  Item(s)
    Item

Alternation

Alternation is represented with the IF control word or with the notation (1,0).

IF segment descriptive code = 1
    material
IF segment descriptive code = 2
    supply

This convention is seldom used because the WHILE handles most situations for the case of data structuring.

The other type of alternation is within a string of data items where the item can either exist or not exist. Another way of representing a non-required item.

1. item-1
2. item-2
3. item-3(1,0)

This says that items 1 and 2 must exits or are required and item 3 is optional.

Discussion

The highest level of repetition within a data structure is assumed to be the key to the file or at least the major sort sequence. If additional keys are required they can be represented with the word KEY [i.e., item-3 (KEY)] or an additional data structure can be presented to represent the structure represented when the KEY is used.

GOALS can be used to represent logical structures as well as the physical implementations. It is important that the required logical views of data be derrived and documented before any physical structures be planned. A goal in system design is to have a one to one relationship between the physical and the logical structures of the system. The coding complexity is reduced appreciably as wellas the maintainence activity. An additional byproduct is the ability to use Query or other general inquiry languages in a more straight forward fashion.

LEVELS: are represented graphically with the use of indentation. The first character in a line is considered to begin an "A" level subsequent levels are indented an additional three spaces each.

Successively lower levels (higher value characters and more deeply indented) represent subordinate processes. As will be seen in the general system structure the highest most levels are controled by increments of time; years, quarters, months, days, etc. while lower levels are controlled by events or conditions.

CONTROL NUMBERS: The control numbers used in GOALS are developed by alternating the use of numbers and letters to represent sucessively lower levels within the system. The system is similiar to English outlining except that only capital letters and numeric characters are used. For a given statement there is nothing to indicate its position in the hierarchy unless the entire control number is dipicted or the starting control number on the page is given. When GOALS statements are machine stored the entire control number is either stored or is assumed.

Principles

An Information system is distinguished from operating systems, command interperters, compilers and the like. An Information System is that set of communications, operations, files and outputs associated with a single conceptual "file."

I am not talking about a single program. Historically I am talking more about an application area.

Elements

Components

First an analogy: All purely mecanical devices are made up of elemental components; the incline plane, the wheel and axle, the lever and the chamber. The physics of these basic components and the materials from which they are constructed define the limits of their application. You may be saying, that list does not sound correct or "what about the screw." In listing elemental components certain definitions are inherent. I define the screw as a "rolled incline plane."

For information systems I assert that the list is: Communications, files, operations and outputs. The limits for such systems are defined by the ordering of the elements using the primitive structures (sequence, alternation and repetition).

As a note; to date the list of elemental components may have been input, process and output without regard to structure. This is more elemental considering all computer processes but is unbounded. This makes a general system design technique very difficult. Adding hierarchy to the above does not enhance these primitives to any great extent.

Relationships

With these boundries and definitions in hand, lets look at the relationships that develop.

There is generally a one to one relationship between file structure and operations structure, between communications structure and operations structure, between output structure and operations structure. In other words the operations or control structure mimics the other components of the system and each componet is related to the other in structure. Structure begins with the file structure.

Example; if you have a file of accounts and you want

to report them; the report program may need to be structured exactly the same as the file or database to report all the data in the file. Most often there is a one to one relationship between files and outputs. In the report example the report structure could be expected to look exactly like the file. If the report is to look different than the file there would be in intervening operation usually a sort or selection to convert an intermediate output to the final output.

The same is true of communications which on the data processing level are the transmissions to the uses, the screens and the messages. The structure of a communication is generally the same as the operation structure which is the same as the data structure and thus the communication structure is the same as the data structure. This substantiates the theory that systems can be completly described knowing only the data structure. True but limited. Knowing the structure of any part should in theory give you the whole.

If everything describable about a system can be described in simple structures (and thus in GOALS) and the components of a system include only communications, files, operations, and outputs and GOALS can be used in all system phases then we have a framework for a general system covering conception through maintainence.

Lets look at any application. Traditionally you would begin with a requirements statement and do an analysis of the existing system. Forget flowcharts, classic narratives, and other charting techniques. Think of progressively decomposing the system using simple english outlining starting with the functions. Functions fit into the operations structure discussed. You will note that as you get down a level or two you will encounter repetitive tasks dependant on conditions, add WHILE and IF to your outlines and keep describing. Remember that users can understand outlines and repetition and alternation are not difficult to understand.

Operations will include existing machine processes, manual proceedures, paper flows, sorting processes etc. As you are going through the operations keep a list of the files that are mentioned and note the file keys (and sorts) and any advantages or requests for multiple keys.

List any outputs or reports prepared by the organization or required in the future.

Communications will be minimal at this stage but note any memos that may go from one section to another of a "file" of notes used as crossreference or duplicate of any more perminent file.

Your documentation is now shaping up; your notebook and I assume that the whole world has change to 8½"S11", should be divided into communications, file, operations and outputs.

The starting point for design is the detailing of the files in your file list. You will want to reduce the files as much as possible to a single file. By way of naming conventions the "file" should have the same name as the system at hand.

You will notice that many of the manual files are really communications in that they are "views" of the file that are required in a particular subfunction.

The design of the conceptual file must be validated against the required operations. I am going to leave this hanging for a moment to discuss a General System Structure.

General System Structure

A General System Structure is presented on the following page in Goals.

This structure is not applicable in all systems but is used as a pattern for system discription, design and understanding.

The key elements of design of this structure are:

1. File unity; a system with this structure has only one conceptual file. It may have any number of datasets of or physical files but they must be formalized into one.
2. Journalizing or logging; all changes made to data items can be (and normally should be) logged.
3. Last action dating; incorporated as part of logging, permits an offline log.

One detailed implication of this is need to have a date stamp in each detail set and a master date stamp in the master file.

Note: sleeper from the contributed library is a must. A standard job stream to prepare the system for shutdown and to bring it back up to production mode is also recommended. Allocation of application programs a desirable feature is the reason for this and also a good way to get sleeper going again.

General System Architecture

```
Begin system
WHILE NOT EOSystem
  WHILE NOT EOYear
     WHILE NOT EOQuarter
        WHILE NOT EOMonth
           WHILE NOT EODay
              WHILE ONLINE
                 Begin online
                 identify operator and security
```

```
                    Open system file
                    Open current files
                    WHILE  Communication
                       IF control transfer
                          transfer control
                       IF batch request
                          initiate request
                       IF update , add or delete
                          Begin
                          Memo to LOG
                          LOCK
                          Update ,add or delete
                          UNLOCK
                          End
                       IF inquiry
                          perform communication operation
                       .
                    !
                 !
              !
              End Online
              Begin daily batch
              Perform daily batch processing
              Run LOG analysis
              If end of week
                 Perform Monthly Processing
              .
              ROLL FILES
           !
           perform Monthly processing
        !
        Perform Quarterly processing
     !
     Perform end of year processing
!
Close system
End
```

A GENERAL DESIGN

With this Architecture and database design complete we have the basis for the development and implementation of any application.

Step 1 is inquiry into our file; if there is only one search criteria then we calculate into to file and return the master data or a summary. Once positioned in a master we can chain through our detail sets or follow appropriate programatic paths.

The master screen (a communication) should provide inquiry, update, and addition ability.

Each detail set should have a screen providing the same update add and inquiry ability. Our screens will be one for one with the detail sets. Think of a detail set as having a buffer that will correspond to communication (VPLUS) buffer. Moving data within one program is facilitated with this concept.

The list of detail sets becomes a list of programs which must be written to handle the retrieval, update, addition, deletion and editing of data for the detail set.

When this is complete you will have a functioning system; it will not function well. I have intentionally oversimplified. The office proceedures which may be in place or will evolve will dictate what combination of sets will appear on a screen but no effort was be lost in developing the barebones system according to this method. Each set (detail set) should have its own program to handle retrieval and update. When requirements demand inclusion the programs can usually be used with few changes. You can take this one step further to include a general scheme to handle multiple data sets on one screen.

The question then becomes; "How do I tie this all together?"

Interactivity and Control

Let's say that we have written a system composed of a series of programs that correspond to our data sets. The way in which we implement interactivity is through a control program called MENU. 4A Menus

A master data set will exist at the top of the conceptual file and the primary search path will be the file key. Other search paths will be provided through subsystems such as "Name Family" or through automatic masters. For all detail sets associated to the master there will be

a program to handle that data set. Your analysis will dictate all the processes that the operator may wish to perform.

As other requirements develop associating more than one data set the code can be combined and new screens developed.

The menu control program provides transfer of control. It can do this either "quietly" or "loud." Loud is the obvious implementation; the operator choses a data set from a menu screen, the control is transferred via a "call" to a dynamic subprogram the data set is accessed updated, etc. and control returns to the controlling menu. But let us give the operator the ability to "tell" the system where he wants to go next. If he does a common area flag can be set to say don't display the menu simply transfer control to some other subprogram. We call are common area for data SYSBLK and out flag(s) Q1, Q2, etc. (you are not limited to one level of menu).

A menu structure may look like this:

```
    MAIN MENU
WHILE NOT PARENT OR END OF SYSTEM
  IF LOUD
     GET MAIN MENU SCREEN
     SEND (SHOW) SCREEN
     WHILE  EDITS'FAILED
       EDIT FIELD
       IF EDITS FAIL
          SEND SCREEN
       .
     SET MODE TO QUIET
  IF QUIET
     IF NEXTPROCEEDURE=A
        CALL A
     IF NEXTPROCEEDURE=B
        CALL B
          ...
     IF NEXTPROCEEDURE =N
        CALL N
     ELSE
        CALL CONTROL'NUMBER'TABLE
     .
  !
```

Through this technique those programs which are not being used are not using memory resources. The CONTROL NUMBER TABLE refers to implementations which have levels of menus. If the control reference is not handled at that menu level control is appropriately passed to the proper level where a control program can handle it.

The quiet "CALL" technique can be used for any of the data set programs discussed by putting the quiet call structure "around" the program and requiring the passing of appropriate data into or from the communication buffer. If you need to pass data from one subprogram to another and you want to release the calling program stack space you can do so with extra data segments (DMOVIN, DMOVOUT) or message files or scroll files (logical device dependant files) that you set up in the application program i.e. BUILD SCROL033;rec=-80,16,f,ascii.

Pitch for the use of intrinsics; we have found that most 3000 users do not take advantage of some of the very rich intrinsics in MPE. They are simple calls, well documented and even those that require bit settings are fairly easy to implement in any language.

The COMMAND intrinsic, for example lets you issue MPE command line, commands programatically. We use this to create stream jobs then kick off the job from online programs. A report menu can be used this way.

Effect of called programs on the Stack

The effect of using properly implemented called programs is simple and dramatic. You reduce the amount of stack (that normally translates into main memory) that is required by each user of an application program. Jim Kramer HP SE Saint Louis (Quad Editor Fame) calls it timesharing the stack.

Usually the outerblock program carves out the required amount of data area to be shared by all subprograms in the "system"; this would normally include a database area, a VPLUS area and an area for the system at hand. MPE then carves out some data area for Image and VPLUS. Using a simple menu concept as discussed, as each program is called it will require its own data area and thus addition stack on top of the common (Q relative) data area, when the program returns to the menu this stack space will be unused but as soon as the next program is called this same space will be used by that program for its space.

COBOL sections do not do the same thing. They create data areas for all declared data in the data division. Sectioning permits smaller code segmentation but this is a shared resource on the 3000 anyway. Note that with stack sharing per user that the reduction in memory requirements is greatly enhanced over code optimization.

You will also find that editing code is much easier with smaller source files, that compilation is faster and more concise code is written.

SL's and USL's

SL's

- Modules, entry points and called Programs require 1 CST entries if they are not already referenced in a running process.
- Code is sharable by all programs. The PUB.SYS SL is avalable to all programs. Account and group SL's are available to programs being run out of that Account or group.
- You may exclusive access to the SL to make an entry in it.
- When SL entries are made you do not need to prepare the SL. It is available after you have exited the segmenter.

USL's

- Programs compiled into a USL must be prepared before they are runnable.
- Many programs may be compiled into the same USL. When a program is run the system will look to the USL for resolution of called programs, it then looks to the PUB.SYS SL unless a library is specified in the RUN. (RUN prog;LIB=G)

- All USL resolved entries create XCST entries except the outer block.

CST's and XCST's

- There are 192 CST entries available to user processes
- There are 1028 XCST entries available to user processes.

COMPILE INTO A USL COBOL/3000 Example

```
!JOB JOBNAME,username/userpass.accountname/accountpass;OUTCLASS=
!COBOL progname,$NEWPASS,$NULL
!SEGMENTER
USL $OLDPASS                    **  only needed  **
NEWSEG progname,progname'       **      for      **
PURGERBM SEGMENT,progname'      **  COBOL/3000   **
USL yourusl
PURGERBM SEGMENT,progname
AUXUSL $OLDPASS
COPY SEGMENT,progname
EXIT
!TELL user.acct; yourprog ---> yourusl
!EOJ
```

PREP OF USL

```
!JOB DyourUSL,user/userpass.account/accountpass;PRI=ES;OUTCLASS=
!PURGE yourrun
!CONTINUE
!BUILD yourrun;DISC=2500,1,1;CODE=PROG
!SEGMENTER
USL yourusl
PREPARE yourrun;MAXDATA=16000;CAP=MR,DS
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

CALLABLES INTO SL's

```
!JOB D!SL,user/userpass.account/accountpass;OUTCLASS=,1
!COBOL yourprog,$OLDPASS,$NULL
!SEGMENTER
AUXUSL $OLDPASS
SL SL
ADDSL yourprog
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

MENU

```
REPEAT until parent or end of system
   IF loud
      get menu screen
      show screen
      REPEAT until edits pass
         edit fields
         IF edit fail
             send screen
      !  .
      set mode to quiet
```

```
.
IF quiet
   IF nextprocedure = "O"
       CALL "O" USING ., ., .
   IF nextprocedure = "I"
       CALL "I" USING ., ., .
        .
        .
        .
   IF nextprocedure = "n"
       CALL "n" using ., ., .
   ELSE
       CALL "CONTROLNUMBERTABLE" using nextprocedure
   .
!  .
```

Goals-SPL Standards

Section	Title
1	General
2	Procedures and Declarations
3	Moves
4	IF Control
5	REPEAT Control
6	Witan include files
7	Coding rules

GOALS-SPL STANDARDS

General

Indentation of three spaces indicates the beginning of a new level. If the next line is indented six spaces it indicates a continuation of the previous line.

Assignment is done with the ":=."

Comparison is done with the "=".

The astrisk is used to indicate that the address required in a statement has already been loaded on the stack. This has general applicability but we will limit its use to moves where the previous move has used the stack decrement option leaving the ending address on TOS. In a MOVE WHILE there is a stack decrement feature, a ",1" following the A, AN or N indicates that the final destination address is left on TOS.

The asterisk in parenthesis (*) indicates a backreference to another data item causing a redefinition of the area in the data stack. This back reference does allocate one word of the stack as a pointer.

Parameters should always be on word boundries thus BYTE ARRAYS should not be used as parameters.

Procedures and Declarations

Procedures parameters should all be called by reference not by value.

The form for an outer block program is:

```
$CONTROL USLINIT [ ERRORS=5, LIST, ... ]
BEGIN   << SOURCE >>
  [global data declarations]
  [procedures/intrinsics]
  [global-subroutines]
  [main-body]
END. <<  SOURCE  >>
```

The form for a subprogram is:

```
$CONTROL SUBPROGRAM [ERRORS=5, LIST, ... ]
BEGIN  << SOURCE >>
  [compile time constructs]
  [procedures/intrinsics]
END.   << SOURCE >>
```

The form of a sample subprogram using the Witan INCLUDE files found in the appendix follows:

```
$CONTROL SUBPROGRAM,ERRORS=5,NOLIST,NOWARN,SEGMENT=SEGNAM
BEGIN << SOURCE >>
$INCLUDE INC1G.T

<< BEGIN EXTERNAL PROCEDURE DECLARATIONS >>
$INCLUDE STDINTR.T << STANDARD EXTERNAL PROCEDURE DECLARATIONS >
PROCEDURE BLANK(WINDOW,VI);
   VALUE VI;
   IA WINDOW;
   IN VI;
   OPTION EXTERNAL;
<< END EXTERNAL PROCEDURE DECLARATIONS >>

PROCEDURE SEGNAM(VBLK,SYSBLK,RTN'CDE);
   IA VBLK,SYSBLK;
   IN RTN'CDE;
BEGIN << SEGNAM >>

<< BEGIN DATA >>
$INCLUDE VBLK.T
$INCLUDE SYSBLK.T
IA IBLK(0:0);
$INCLUDE SUBGLOB.T     << USING SUBGLOB.T REQUIRES THAT VBLK,IBLK
                          SYSBLK HAVE BEEN INCLUDED IN THIS PROCE
                          EITHER AS PASSED PARAMETERS OR AS NULL
                          ARRAYS. >>
<< OTHER DATA LOCAL TO PROCEDURE >>
LG KEEP'GOIN;
IN VI;
IN MISC;
IA (0:9)TEN'WORDS;
<< END DATA >>

<< BEGIN SUBROUTINES >>
SUBROUTINE PUT'WINDOW;
   BEGIN << PUT'WINDOW >>
      V'PUT'PAUSE(VBLK,2);
      BLANK(WINDOW,30);
      WINDOW'LEN:=60;
      VPUTWINDOW(VBLK,WINDOW'LINE,WINDOW'LEN);
      VSHOWFORM(VBLK);
   END; << PUT'WINDOW >>
<< END SUBROUTINES >>

<<**********************************>>
BEGIN  << CODE >>

KEEP'GOIN:=TRUE;
WHILE        KEEP'GOIN DB
   KEEP'GOIN:=FALSE;
END'REP;

END; << CODE >>
END; << SEGNAM >>
END; << SOURCE >>
.
```

Moves

General Forms:

MOVE destination:=source, (length)[,stack decrement];

Literals:

Length need not be specified in the move of a literal If successive moves are anticipated to build a string or concatenate into a buffer then the stack decrement option of 2 can be used. Example:

```
MOVE OUTBUF:= "Hello",2
MOVE          *;=" Everyone";
```

Non-Literals:

SPL requires type compatibility in moves, therefore general buffers should be defined in words and in bytes. The word buffer name should end with "'W." The byte buffers will have the just name without an identifying sufix.

The length parameter in the move should specify a name equated to the length in bytes or words depending on the type of move. The equate will generally be generated by DBUF. Byte lengths will begin with "BL' ", word lengths with "WL'."

Example:

```
MOVE OUTBUF:=
   ACCOUNTNO,(BL'ACCOUNTNO);
```

Some moves may embed procedure to insure type compatibility and at the same time perform the appropriate conversion.

IF Control

The control structure for the IF will follow directly the structure enforced in GOALS. All IF's will be followed by a condition which may be compound and may extend to subsequent lines (note; continuation line disciple in general standards).

Following an "IF" condition a TB will be inserted, which is defined as a "THEN BEGIN." SPL does not require a BEGIN if the following statements are not compound, i.e., a lone statement. However, the "BEGIN" is required to bracket the sequence and to enforce the use of an "END" on the same level as the beginning "IF." If there are subsequent "IF's" on the same level (mutually exclusive IF's — programmer enforced) the IF should be converted to an IF'G which is defined in INC1G as an "END ELSE IF." This is not called a "IF" in GOALS. It is refered to as an "IF string" (mutually exclusive conditions).

Nested IF's:

If "IF's" are nested, the nested IF may begin any time after the "TB" of the preceding IF and will be indented to show its nesting. The rules for the nested IF are exactly the same as the IF; TB required.

ELSE

When the trailing ELSE is required in an IF string, the preceding end for the IF must not have a semicolon. The ELSE requires a BEGIN-END pair to enforce the terminating "END" at the end of the IF string.

Nested IF strings, where trailing elses come together may cause some confusion, but do not require any special rules.

Example:

```
IF --condition--          TB   << THEN BEGIN >>
  IF --condition--        TB
     --statm't--;
     --statm't--;
  IF'G --condition--        TB
     --statm't--;
     --statm't--;
  ELSE'G
     --statm't--
  END'IF;
ELSE'G
  --statm't--;
  --statm't--;
END'IF;
```

Repeat

General Form:

```
WHILE          --condition--          DB
   --statm't--------)
   --statm't---------
END'REP;
```

The REPEAT in the GOALS-SPL is used as documentation and is defined as a null statment. REPEAT must be followed by WHILE and a condition or compound condition. Following a WHILE condition a "DB" is required which is DEFINED in INC1G as a "DO BEGIN." As in the IF construct a "BEGIN" is required to enforce a terminating "END'REP."

```
BYTE POINTER
   BP  << USED FOR TEMPORARY POINTER, NOT SAVED >>
  ;
EQUATE
   RTN = 13  << CARRIAGE RETURN IN ASCCI >>
  ,ESC = 27  << ESCAPE CHARACTER IN ASCII >>
  ;
INTEGER I,J,K,LEN80,OLD'LANGUAGE;

  DA IBLK'D          (*) =  IBLK;
  BA IBLK'B          (*) =  IBLK;

  DA SYSBLK'D        (*) =  SYSBLK;
  BA SYSBLK'B        (*) =  SYSBLK;

  DA VBLK'D          (*) =  VBLK;
  BA VBLK'B          (*) =  VBLK;
DEFINE
   EL = END ELSE#
  ,END'IF = END#
  ,END'REP = END#
  ;
```

INC1G.T

This INCLUDE is used for abbreviation of data types and some constructs for GOALS presentation SPL compilations.

```
DEFINE <<USED TO ABBREVIATE DATA TYPES>>
   IA = INTEGER ARRAY#
  ,IN = INTEGER#
  ,DI = DOUBLE #
  ,LA = LOGICAL ARRAY#
  ,DA = DOUBLE  ARRAY#
  ,BA = BYTE    ARRAY#
  ,RA = REAL    ARRAY#
  ,XA = LONG    ARRAY#
  ,LP = LOGICAL PROCEDURE#
  ,DB = DO BEGIN#
  ,TB = THEN BEGIN#
  ,LG = LOGICAL#
  ,REPEAT = #
  ,G'IF = END ELSE IF#
  ,G'ELSE = END ELSE BEGIN#
  ,IF'G = END ELSE IF#
  ,ELSE'G = END ELSE BEGIN#
  ;
```

IBLK.T

```
<< IA IBLK(0:42); >>
  DEFINE
     COND'WORD        = IBLK #,
     STAT2            = IBLK(1) #,
     STAT3'4          = IBLK'D(1) #,
     STAT5'6          = IBLK'D(2) #,
     STAT7'8          = IBLK'D(3) #,
     STAT9'10         = IBLK'D(4) #,
     BASE             = IBLK(10) #,
     MODE1            = IBLK(26) #,
     MODE2            = IBLK(27) #,
     MODE3            = IBLK(28) #,
     MODE4            = IBLK(29) #,
     MODE5            = IBLK(30) #,
     MODE6            = IBLK(31) #,
     MODE7            = IBLK(32) #,
     MODE8            = IBLK(33) #,
     ALL'ITEMS        = IBLK(34) #,
     PREV'LIST        = IBLK(35) #,
     NULL'LIST        = IBLK(36) #,
     DUM'ARG          = IBLK(37) #,
     NUM'BASE         = IBLK(38) #,
     IBLK'LEN         = 43 #
     ;
```

IBLKG.T

The following is initilization code to be included in the outer block program to set IBLK fields :

```
MODE1              := 1;
MODE2              := 2;
MODE3              := 3;
MODE4              := 4;
MODE5              := 5;
MODE6              := 6;
MODE7              := 7;
MOVE ALL'ITEMS     := "@;";
MOVE PREV'LIST     := "*;";
MOVE NULL'LIST     := "0;";
DUM'ARG            := 0;
```

VBLK.T

```
<<   THIS ASSUMES THAT VBLK IS DECLARED IA VBLK(0:51)          >
<<     VBLK IS MADE UP OF COMAREA AND THE OLD VBLK             >
<<  CALLS TO VIEW INTRINSICS WILL USE VBLK AS THE COMAREA PARM >

   <<SPL DECLARATIONS FOR COMAREA>>
   DEFINE
      COM'STATUS          = VBLK (0)   #,
      COM'LANGUAGE        = VBLK (1)   #,
      COM'COMAREALEN      = VBLK (2)   #,
      COM'USRBUFLEN       = VBLK (3)   #,
      COM'CMODE           = VBLK (4)   #,
      COM'LASTKEY         = VBLK (5)   #,
      COM'NUMERRS         = VBLK (6)   #,
      COM'WINDOWENH       = VBLK (7)   #,
      COM'LABELSOK        = VBLK (9)   #,
      COM'CFNAME          = VBLK'B (10*2)  #,
      COM'NFNAME          = VBLK'B (18*2)  #,
      COM'REPEATAPP       = VBLK (26) #,
      COM'REPEATOPT       = VBLK (26) #,
      COM'FREEZAPP        = VBLK (27) #,
      COM'CFNUMLINES      = VBLK (28) #,
      COM'DBUFLEN         = VBLK (29) #,
      COM'DELETEFLAG      = VBLK (32) #,
      COM'SHOWCONTROL     = VBLK (33) #,
      COM'PRINTFILNUM     = VBLK (35) #,
      COM'FILERRNUM       = VBLK (36) #,
      COM'ERRFILNUM       = VBLK (37) #,
      COM'FM'STORE'SIZE   = VBLK (39) #,
      COM'NUMRECS         = VBLK'D (21)    #,
      COM'RECNUM          = VBLK'D (22)    #,
      COM'TERMFILENUM     = VBLK (48) #,
      COM'TERMMODE        = VBLK (49) #,
      COM'TERMALLOC       = VBLK (50) #,
      COM'DATAOVERRUN     = VBLK (51) #,
      COM'READTIMEOUT     = VBLK (52) #,
      COM'OTHERDATAERR    = VBLK (53) #,
      COM'MAXRETRIES      = VBLK (54) #,
      COM'TERMCONTROLOPT= VBLK (55) #,
      COM'TERMOPTIONS     = VBLK (55) #,
      COM'ENVINFO         = VBLK (56) #,
      COM'TIMEOUT         = VBLK (57) #
   ;
   EQUATE
      COMAREALEN          = 60,
      COBOL'LANG          = 0,
      VBLKLEN             = 100,
      SPL'LANG            = 3,
      MAXWINDOWLEN        = 150,
      MAXMODELEN          = 8,
      NAMELEN             = 15,
      NORM                = 0,
      NOREPEAT            = 0,
```

```
      V'REPEAT          = 1,
      REPEATAPP         = 2,
      ENTERKEY          = 0,
      PARENTKEY         = 1,
      KEY2              = 2,
      KEY3              = 3,
      REFRESH           = 4,
      PREV              = 5,
      NEXTKEY           = 6,
      INQ'ENT           = 7,
      EXITKEY           = 8
   ;
   <<SPL DEFINITIONS FOR VBLK>>
   DEFINE
      WINDOW'LEN        = VBLK (COMAREALEN+0)     #,
      MODE'LEN          = VBLK (COMAREALEN+1)     #,
      WINDOW'LINE       = VBLK (COMAREALEN+2)     #,
      WINDOW'MODE       = VBLK (COMAREALEN+2)     #,
      WINDOW            = VBLK (COMAREALEN+MAXMODELEN)     #,
      WINDOW'LINE'B     = VBLK'B ((COMAREALEN+2)*2) #,
      WINDOW'MODE'B     = VBLK'B ((COMAREALEN+2)*2) #,
      WINDOW'B          = VBLK'B ((COMAREALEN+MAXMODELEN)*2) #
   ;
```

SYSBLK.T

```
<<IA SYSBLK(0:114)  SPACE ALLOCATED IN MAIN PROGRAM >>
DEFINE
    CNTRL'NUM       = SYSBLK #,
    LST'PROC        = SYSBLK(2) #,
    NXT'PROC        = SYSBLK(4) #,
    Q1              = SYSBLK(6) #,
    Q2              = SYSBLK(7) #,
    Q'NEXT          = SYSBLK(8) #,
    OPER'ID         = SYSBLK(9) #,
    SECU'TY         = SYSBLK(11) #,
    SSC             = SYSBLK(13) #,
    CNUM            = SYSBLK(16) #,
    L'FLNUM         = SYSBLK(21) #,
    M'FLNUM         = SYSBLK(22) #,
    FLAGS           = SYSBLK(23) #,
    DQSTAT'SB       = SYSBLK(28) #,
    GLSTAT'SB       = SYSBLK(43) #,
    TERMID          = SYSBLK(58) #,
    MSBLK'SB        = SYSBLK(63)#  << STARTING ON DOUBLE BOUNDRY
    ;
DEFINE
    CNTRL'NUM'B     = SYSBLK'B #,
    LST'PROC'B      = SYSBLK'B(2*2) #,
    NXT'PROC'B      = SYSBLK'B(2*4) #,
    OPER'ID'B       = SYSBLK'B(2*9) #,
    SECU'TY'B       = SYSBLK'B(2*11) #,
    SSC'B           = SYSBLK'B(2*13) #,
    CNUM'B          = SYSBLK'B(2*16) #,
    FLAGS'B         = SYSBLK'B(2*23) #
    ;
EQUATE
    CNTRL'NUM'BL    = 4,
    LST'PROC'BL     = 4,
    NXT'PROC'BL     = 4,
    OPER'ID'BL      = 4,
    SECU'TY'BL      = 4,
    SSC'BL          = 6,
```

```
CNUM'BL          = 10,
FLAGS'BL         = 10
;
```

Coding Rules

All agorithms should first be done in GOALS without concern for the SPL structure. SPL constructs will be used for individual statements and conditions but the control structure should be in GOALS.

This complete:

1. Replace all ELSE's with G'ELSEs or ELSE'Gs.
2. Locate all "IF's that are on the same level as a "running" IF. Replace each running IF with an IF'G or G'IF.
3. Replace all "."'s with an END'IF;
4. Insert a THEN BEGIN or "TB" following every IF condition.
5. Replace all "!" with an END'REP;.
6. Insert a "DB" or DO BEGIN after every REPEAT condition.

An Example using the rules on the preceeding page

```
WHILE          ------------
   IF ------------
      --------------
      --------------
      IF ------------
         --------------
      ELSE
         --------------
         --------------
      .
   IF ------------
      --------------
   .
   IF ------------
      --------------
   IF ------------
      --------------
   ELSE
      --------------
   .
!
```

```
                                          <<   SPL      RULE >>
WHILE          ------------          DB   << DO BEGIN     6  >>
   IF ------------                   TB   << THEN BEGIN   4  >>
      --------------
      --------------
      IF ------------                 TB   << THEN BEGIN   4  >>
         --------------
      ELSE'G                               << END ELSE BEGIN  1 >>
         --------------
         --------------
      END'IF;                              << END'IF        2  >>
 ***(20) ERROR ***
LINE
1490
 TRUNCATED BY  4 CHARACTER(S)
      IF'G ---------                   TB   << END ELSE 2 >>  << THEN B
         --------------
      END'IF;                              << END'IF        3  >>
      IF ------------                 TB   << THEN BEGIN   4  >>
         --------------
 ***(20) ERROR ***
LINE
1495
 TRUNCATED BY  4 CHARACTER(S)
```

```
   IF'G ---------                TB   << END ELSE 2 >>  << THEN B
     --------------
   ELSE'G                             << END ELSE BEGIN  1 >>
     --------------
   END'IF;                            << END'IF      3  >>
END'REP;                              << END'REP     5  >>
```

Note: work the top example yourself using the rules and see if it matches the completed program. Note the count of the begins and ends match for SPL. Do the algorithm correctly in GOALS and the SPL code will follow.

Power Line Disturbances And Their Effect On Computer Design and Performance

Vince Roland

The following is extracted from an article in the August 1981 Hewlett Packard Journal, by Vince Roland and Art Duell.

One of the earliest and continuing problems with computer systems is ac power line disturbances on customer premises. The computer manufacturer is becoming increasingly concerned about the ac line transient and grounding environment that a computer system is subjected to at a customer site.

The resolution of problems caused by ac power disturbances requires characterization of the ac power source and determination of the computer system's sensitivity to such anomalies. The effect of any possible solution on the manufacturer and the customer must be evaluated. If the manufacturer incorporates devices to protect the computer against all of the possible ac disturbances the initial cost of the computer increases significantly. A dilemma arises if, to keep purchase costs low, this is not done. In the purchase of any computer system, the cost of ownership must also be considered. The customer wants and should expect maximum use with a minimum of maintenance and downtime. If the computer is unable to handle power anomalies, the downtime and maintenance can become excessive and the cost of ownership increases dramatically.

Therefore, some compromise is required to minimize the overall cost to the customer. The customer should help by improving the environment for the computer system installation. The manufacturer should correctly and economically specify the environment required and educate the customer about this specification in addition to incorporating economical design features that improve the computer system's resistance to ac power line disturbances.

CHARACTERIZATION OF POWER SOURCES

Improving the immunity of a computer system to electrical noise requires adequate characterization of the ac power source. The terms frequently used to describe power line anomalies are discussed at the back of this paper. The noise present on an ac line can be generated by conditions unique to the customer's environment and by variations typically found on any power line supplied by a public utility. Wiring codes developed by regulatory agencies to insure the safety of the user are often in conflict with line configurations designed for noise reduction.

Computer manufacturers and users cannot influence or change these facts significantly. To place these conditions in proper perspective, note that statistics show that computer systems and other sensitive control loads represent less than 0.01% of the total utility load. Consequently, it is understandable that a utility will not try to prevent power disturbances affecting computers. Wiring codes are also generated with respect to the general consumer. Generally wire sizes and grounding specifications are based on electrical loads associated with major appliances and heavy electrical machinery, and do not take into consideration the low impedances that are required when computer systems are switching at millisecond speeds.

Factors describing the quality of an ac power line are nominal line voltage, service voltage, utilization voltage, statistical distribution of transients generated by the utility and the customers, and regulatory body specifications. In the U.S.A. there are national standards developed by American National Standards Institute (ANSI), and these specifications are typically used by U.S. utilities to permit better networking and interchange of power.

Nominal line voltage is the level close to the average value expected during normal operation. This value, measured at the outlet, varies with geographic location.

The most important parameter that the utility must adhere to is the service voltage — the voltage supplied to the customer's meter. In the San Francisco Bay area it is 114 volts minimum to 126 volts maximum. For three-phase-wye distribution, it is 197 volts minimum to 218 volts maximum. Generally speaking, the three-phase line voltage will be lower than the nominal 208 volts, but the power company is still within specifications as long as the line voltge does not go below the minimum service voltage.

The utilization voltage (at the wall outlet) is 110 volts minimum to 125 volts maximum. For three-phase-wye distribution, it is 191 volts minimum to 216 volts maximum. Herein lies a difficulty. The building owner is responsible for all internal wiring within the building from the meter to the wall outlet. Therefore, if there is a

problem with the voltage, it is of little value to a user to have the power company measure voltage at the wall outlet because they have no control over the internal wiring of a building.

To compound the problem, there are some studies being made by U.S. utilities to change generating voltages to conserve energy. The federal government is strongly suggesting to the utilities that there be a conservation voltage reduction. This is a systematic lowering of distribution voltages to reduce energy consumption by customers. The minimum service voltage would remain the same but the maximum would be lowered. Suggested ranges would be either 114 to 118 volts or 114 to 120 volts. When the generating margin narrows, the utilities are forced to do more load switching. The result is more transients.

It is impossible to quantize transients caused by utilities switching loads during peak demand hours, or by breaker action during some fault condition. A model can be developed, but it is a function of such parameters as line impedance, circuit breaker size, fault current, and other widely varying factors. Obviously, the computer installer and customer should be fully aware of possible problems when the computer system is installed near a utility substation or switchyard.

CUSTOMER SITE CHARACTERISTICS

Noise on the power line can be generated in many different ways at the computer user's site. Electric clock systems signal-modulate the power distribution within a facility once an hour to update all electric clocks. Flicker (momentary voltage dip due to the starting of a large appliance) can occur typically 10 times per hour and the duration can be from 160 to 670 ms. The maximum amplitude of a transient is directly proportional to the velocity with which a contact opens, and is independent of power consumption. a 400-hp motor (with large slow-moving contacts) produces transients with one-tenth the amplitude of those produced by an electric clock motor. Fluorescent light switching can cause an extended transient of 2-MHz, 500V oscillations lasting for 20 microseconds. Power characteristics of an installation change with time even though good site preparation is done initially. Vending or copy machines can be added inadvertently to circuits and grounding that were initially wired exclusively for computer systems.

The most common customer problems associated with ac power are nonisolated grounds and improper conductor sizes. These occur even when the grounding and wiring are done according to code.

Two categories of ground systems must be considered:

1. Safety (dc conduction) — the electrical power grounding system which includes all ac power, distribution and utility service power used for lighting, equipment power, et cetera.
2. EMI (RF conduction) — signal circuit grounding which includes all electronic and electrical control circuits associated with a computer system.

In the first category, all neutral and ground line distributions are wired on separate buses and connected together at the main power transformer entrance to the building, making a single-point ground.[1] In the second category, the ground from the electronic equipment is connected to the nearest steel structural beam to make a multipoint ground (see Figure 1) from a facility viewpoint.

Because computer systems typically use earth ground as a reference within each cabinet and the entire system is connected to the facility earth ground, it is very important that the computer system be connected to an EMI grounding system. Figure 2 shows the system power and interface cable hookups and points where common-mode noise (voltage between both lines and ground) can exist. The net voltage difference between any two points on the ground network usually will be small (1 to 3 volts). However, the current through certain network paths can be on the order of 3 to 5 amperes with occasional currents of 10 to 15 amperes.

For typical building wiring, electricians use water pipes and conduit as ground. For safety and minimal shock hazard, this is legal from an electrical code viewpoint. For EMI suppression it is inadequate because lighting loads, vending machines, and other types of office equipment are connected to the same grounding system. Another common contributor to stray ground currents within a facility is the connection of the ac neutral line to earth ground inside the branch panel. Then the ground network becomes a part of the ac return line to the main building service entrance. The line current will divide between the neutral line and the ground return network in inverse proportion to the impedance of the two paths.

Because instantaneous power surges are required by a computer system during turn-on and normal operation, wire sizes must be large enough to keep the voltage from sagging. For example, during computer turn-on, switching power supplies can require currents peaking at 150A and decaying exponentially to 20A in less than 30 ms. If the wire size is inadequate, the input voltage can sag below the required input voltage tolerance of the computer for a period of 100 ms, causing the power-control circuitry to detect a powerfail, thus shutting the system off. Wire sizes specified by typical code requirements are usually at least one size smaller than required for computers. Such code requirements make it difficult for the computer manufacturer to convince the customer and electrician that larger wire sizes must be used if the computer system is to operate satisfactorily.

COMPUTER SYSTEM SENSITIVITY

Circuits used in a modern computer system are extremely fast and more vulnerable to noise than circuits

used a few years ago. Because there is the same high-frequency sensitivity in a peripheral as in the mainframe, the same design parameters are used to immunize the total system from outside disturbances. For software, data integrity is protected from power line transients by using various error correction codes (ECC) in the transmittal of data between parts of a system, and "disc retrys" are used on disc drives when errors occur. Therefore, power line noise can be masked by using software error-correction techniques.

IMPACT ON CUSTOMER AND MANUFACTURER

Making positive determination that a computer installation problem is caused by power line disturbances can be very difficult. The occurrence may be random, and the effects on the system may be different depending on the state of the electronics at the precise time of the disturbance. Symptoms of such problems overlap with those that may be caused by intermittent electrical connections, electrostatic dischange (ESD) either directly to the computer or indirectly via other objects in the immediate vicinity, or even software program bugs. If intermittent ac disturbances are suspected, isolating them may require expensive monitoring equipment that is installed for a period long enough to detect the next occurrence.

The time required to analyze and repair an ac line disturbance problem can be several times the duration required for analysis and repair of other service problems. During the process of diagnosis machine time is lost and the service engineer may have to visit a site several times before proper remedies can be made. In the case of ac power transients, which can be random and are unpredictable, direct correlation of cause and effect is extremely difficult to obtain.

DETECTION OF A NOISE PROBLEM ON-SITE

The service engineer's objective is to prevent power line disturbances from reaching the computer by discerning their characteristics, and then either remove the source or isolate the computer from the source. A comprehensive set of site preparation guidelines is sent to the customer prior to delivery of a computer system. If the guidelines are followed, the likelihood of power line disturbance problems is minimized. The service engineer may participate in site preparation with the customer. Whether during a site preparation or installtion, or in troubleshooting an installed system, a service engineer may proceed through the following steps:

- Look outside the customer's site. Check for major industries in the area that consume large amounts of electricity. Their operation can cause voltage variations or transients to be propagated to other users sharing the same output of the utility company's substation transformer.
- Look within the customer's site for heavy electrical equipment. A transient source within the site may affect the computer installation more severely than a distant source because nearby transients, especially fast-rise-time pulses, do not dissipate significantly in the short distance before they reach the computer.
- Note local weather conditions. Electrical storms may cause transients by direct lightning hits on utility lines or induced coupling through nearby earth strikes. Besides the transients, the utility company's hardware is sometimes affected, causing voltage variations or powerfails.
- Check the wiring from the building's utility power connection to the outlets in the computer room. Feedback from HP's systems specialists in the field indicates that improper site wiring is often the major cause of power line disturbance problems. With the help of an electrician who is aware of local codes, check the building's electrical layout and look for load distributions that overload any circuits, or branch circuits that allow other electrical devices to use the same circuit breaker as the computer. Distribution and breaker panels must have solid electrical connections, and breakers and wire capacities must equal or exceed the computer's demand.
- Check equipment layout at the computer installation. All devices must be plugged into their own wall outlets. Extension cords with multiple outlet boxes must not be used. If possible, avoid extension cords altogether. Check especially for grounding of all devices by having the electrician confirm that the ground wire is continuous back to the building's service entrance. A computer system can pollute its own power if these procedures are not observed.

Up to this point all checks have been visual. If no answers are obvious, measurement equipment is required. Tools for analyzing ac power become progressively more complicated and expensive as the problem becomes more difficult. First, wall outlets can be checked for proper polarity of the lines and ground, and for existence of ground by a receptacle-circuit tester. At the same time, a ground loop impedance tester (GLIT) can be used to test the integrity between the neutral line and ground. However, it only checks impedance at the line frequency, not at high frequency or RF.

When intermittents occur, the cost of the tool goes up significantly and requires the user to have some skill and training in its operation. Such a tool is a power line monitor which can be left at a customer's site for several days and will measure and record voltage surges, sags, frequency variation, powerfails, and transients. Measurements are logged on a printout with their times of occurrence.

Throughout the measurement process, the service engineer notes the nature of the disturbance and checks it against factory-supplied specifications for the computer.

After these checks are completed, a solution is likely to be evident. It may be one of two types: 1) the problem source is identified and can be removed, or 2) the source cannot be removed or cannot be identified, but the nature of the disturbance has been characterized and a device to isolate the computer can be specified.

ISOLATION DEVICES

Isolation devices are available with a variety of features to match the needs of the problem site. Manufacturers offer product lines with varying degrees of protection and power handling capabilities. A qualitative summary of features is given in Table I.

Isolation devices must be installed with full knowledge of their capabilities in mind. Large computer sites set up through subcontractors place no burden of installation (other than financial) on the customer. Customers doing their own installation, however, must work with the service engineer to fulfill the prerequisites before successful operation can happen. Isolation transformers and line conditioners, in particular, are not panaceas that are merely uncrated and connected between the computer and the wall outlet. These devices have their own input and output specifications that must be met, or else a new set of problems will emerge to replace the old ones. All devices in the computer system should be isolated. Otherwise, as shown in Figure 3, an unprotected system component can receive a transient on its ac input and couple the noise to its chassis. Then the noise is coupled to an I/O cable leading to the chassis of the "protected" computer.

ACKNOWLEDGEMENT

I would like to express my sincere thanks to Art Duell of General Systems Division, who co-authored the original manuscript on this topic with me. We both express our thanks to Jim Gillette at the Data Systems Division, whose detailed research and summarizing of various papers, originated new thoughts on test and specification procedures. Larry Rea, formerly of the Customer Engineer Organization, provided valuable inputs based upon experience with solving customer problems, and Jim Brannan in our reliability group has been instrumental in building the customized system-level test tools. Norm Marschke, Computer Systems Division, has made significant contributions in design of system noise immunity and power-control circuits for new HP products.

Table I
Features of ac Line Protection Devices

Device	Description	Voltage Variation Protection	Frequency Variation Protection	Powerfail Protection	Normal-Mode Transient Protection	Common-Mode Transient Protection
Shielded isolation transformer	A transformer with isolated, electrostatically shielded primary and secondary windings.	Input/output ratio can be manually selected by jumpering windings.	None	None	Low. Transformer windings may limit bandwidth, but pulses get through.	High. 120 dB CMR specifications are available.
Tap-switching line conditioner	A shielded isolation transformer regulating output voltage by automatically switching additional secondary windings in or out.	Good. For a broad input range (≈40% tolerance), the output is kept within a ≈15% range.	None	None	Low to medium. Additional filtering may be provided by filter capacitors.	High. 120 dB CMR specifications are available.
Ferroresonant line conditioner	A shielded isolation transformer using a saturated core to clamp voltage to a set-level, and reconstructing the ac sine wave in the secondary.	Good. For a broad input range (≈30% tolerance) the output is kept within a ≈2% range.	None, and will itself malfunction if frequency varies by more than a few Hz.	Low. Energy storage in the core may help if the duration is less than one cycle.	High. Saturated core clamps all pulses. 120 dB NMR specifications are available.	High. 120 dB CMR specifications are available.
Uninterruptible power source (UPS)	Either a motor-generator set, with a diesel engine backup, or a solid-state inverter powered by dc from storage batteries.	High. Alternate power source cuts in if ac line is insufficient.	High. Alternate power source cuts in if ac line is insufficient.	Very good. Duration protection is a function of engine fuel reserve or battery capacity.	Total isolation.	Total isolation.

Definitions: ac Power Anomalies

Power line disturbances may be classified into several types (see Fig. 1):

Voltage variation: The supplied voltage deviates from the prescribed input range. Input below the range is a sag, above is a surge. Sags can be caused by deliberate utility cutbacks (brownouts) to lower power consumption, by customer loads for which the utility cannot compensate, or by an excessive inrush starting current to powered-up equipment. Surges originate from utility line malfunctions or sudden changes in power demand (removal of heavy loads) which cannot be corrected instantaneously.

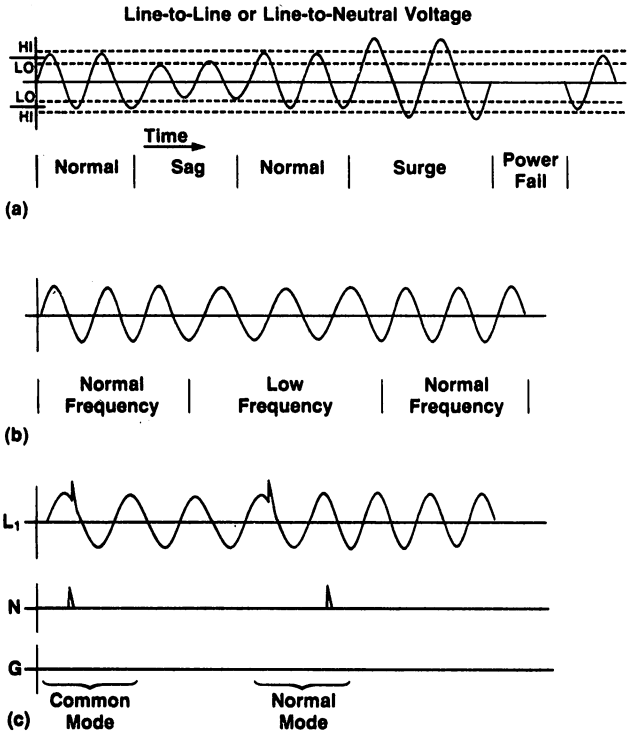


Fig. 1. *Graph of the ac line voltage under (a) normal, sag, surge, and powerfail conditions, (b) normal and abnormal frequency conditions, and (c) with common-mode and normal-mode transients.*

Frequency variation: The frequency of the power line voltage deviates from the prescribed input range. Sudden changes in load to the utility, switching of power between utility companies, or generator malfunction can cause such variations.

Powerfail: Total removal of the input voltage to the computer for at least 5 ms. Switching of power by utilities, either for the purpose of redistributing loads or correcting short circuits, will produce power failures from a few cycles to several seconds. Power equipment failure can result in outages of minutes to hours.

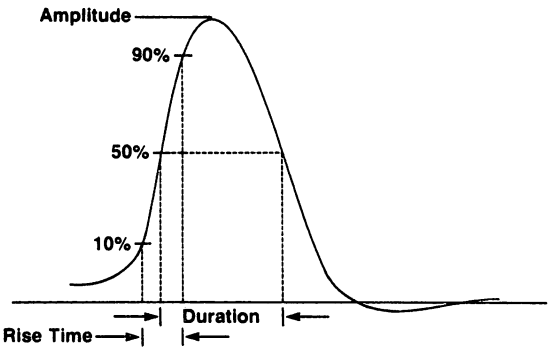


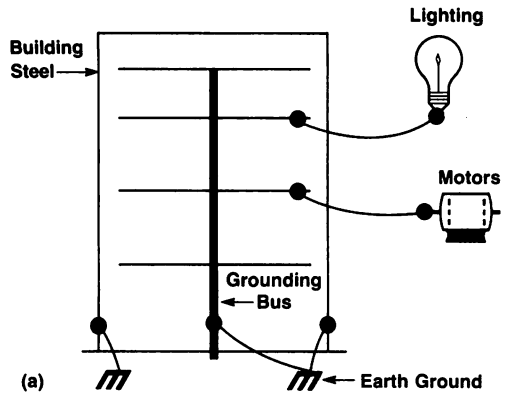
Fig. 2. *Characteristics of transient noise on an ac power line.*

Transient: A disturbance of less than 5 ms duration. The amplitude, rise time, duration, resultant oscillation (if any) and repetition rate (see Fig. 2) determine the effect on the computer's operation. We can classify transients into three types, distinguishing them by their sources.

Transients from nearby sources (within 50 feet of the computer) have very fast rise times (nanoseconds) rich in high-frequency content. The power cord becomes a transmission line, and the propagation of the transient is influenced by distance, conduit, adjacent conductors (into which these transients may be coupled), flatness of the cord against the floor, and socket connections. Because of high-frequency coupling between the conductors in the cord, these transients are usually common-mode by the time they reach the computer. Sources of this type of transient are anything with mechanical breaker contacts, such as coffee pots, electric typewriters, and clock motors.

Transients produced by distant sources will have slower rise times (microseconds) and longer durations than the first type. They are generated by any electrical device that produces enough transient energy to propagate through the device's circuit breaker and distribution panel back to the circuit breaker feeding the computer. Elevator motors, industrial machinery (either on the premises or a block away), and air conditioners are possible sources. These transients are normal-mode or common-mode.

Other transients with rise times similar to those of distant-source transients and with a common-mode structure can be produced by utility distribution faults and resultant arcing, or by lightning, direct or induced, on the utility power pole.



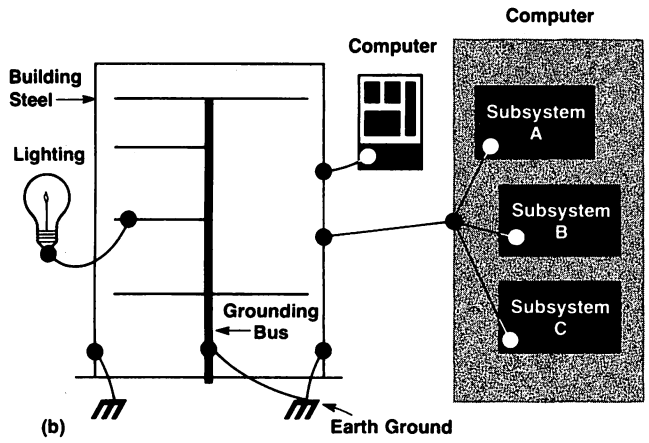


Fig. 1. *The grounding within a building can consist of both (a) single-point (used for lighting, motors, and appliances) and (b) multipoint (used for computer systems) ground bus systems.*

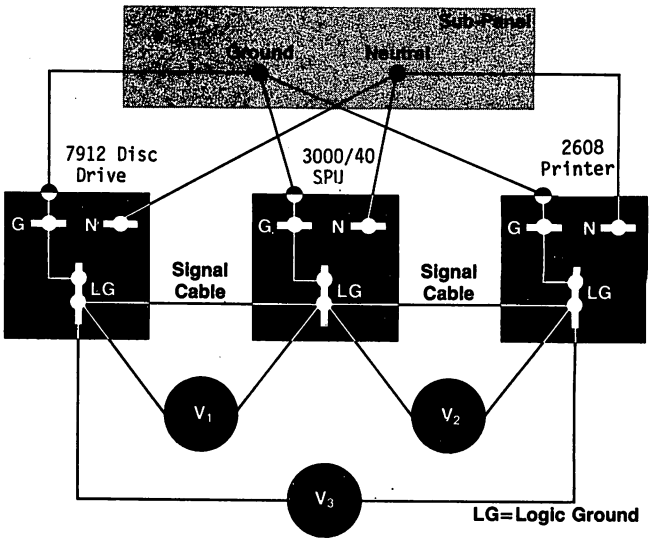


Fig. 2. *Voltage differences between the logic grounds of interconnected computer units can exist if ground impedances and power needs differ between any two units.*

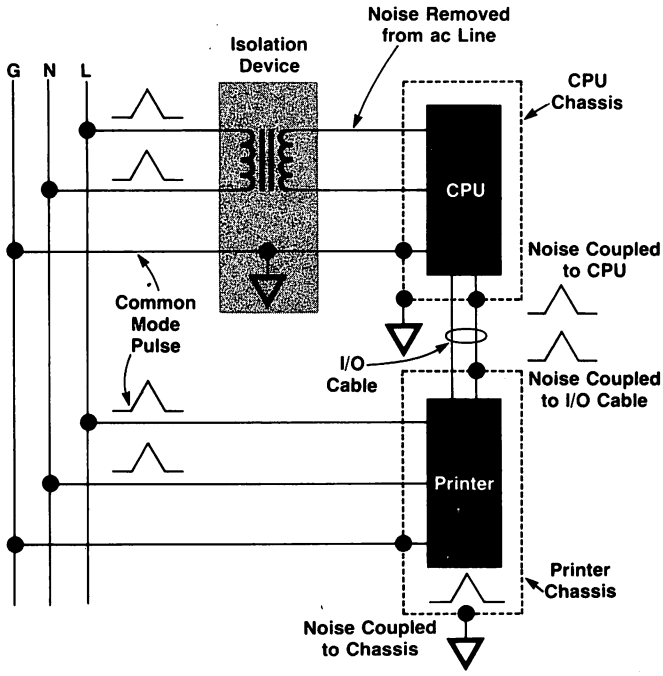


Fig. 3. *All units in a computer system must be isolated from the ac power line. Otherwise, an unisolated unit, the printer shown here, can couple noise to an isolated unit, the CPU here, via the I/O interconnections.*

System Disaster Recovery: Tips and Techniques

Jason M. Goertz
Systems Engineer
Hewlett-Packard
Bellevue, Washington

INTRODUCTION

Since its introduction in 1973, the HP3000 has proved to be one of the most reliable computer systems ever built. Hardware reliability is extremely good, with a minimum of downtime in the case of most users. An extensive field operation exists in the Customer Engineering Organization which, in most cases, can diagnose and repair failing hardware in a very short amount of time. For the really sticky problem that does occur, there exists a large team of engineers in the various manufacturing divisions which backs up the field personnel. This, coupled with a computerized parts inventory system in the field, assures the user that the fastest possible repair will be made to his system. Software reliability is also very good. This is due in a large part to HP's policy that the operating system and subsystems will not be modified by field personnel or by customers, as is the case with many vendors. Along with this, software distribution is handled by the local Field Software Coordinator, giving the field a fair amount of leeway in exactly what software is released to an individual area, while maintaining a reasonable amount of central control.

Unfortunately, all things created by the hand of man are built with imperfections, and this includes computer systems. This paper will deal with the event that these imperfections manifest themselves in such a way as to destroy, or threaten to destroy, the integrity of the system and, of more importance, the data stored on the computer system. This is always more valuable to the owner of the computer than the machine hardware itself.

Causes of this system (and data) integrity loss are many and varied. Usually, a severe hardware failure such as a disc head crash, will cause data to be lost. Many times, a natural occurence (an "Act of God" as the service contract puts it) against which the hardware cannot protect itself will be the culprit. An example might be a severe lightning storm which causes power fluctuations or surges. Software is not free from blame either. Software failures are generally caused by a specific bug which has not been fixed, usually in the operating system. A recent example of this is the PTAPE intrinsic. There were calls to ATTACHIO in PTAPE that were hardcoded to write data to areas in virtual memory on LDEV 1. With the advent of multi-spindle virtual memory on MPE-IV, this became a disasterous situation. A data segment could be built on LDEV 2 or another system domain disc, but PTAPE would write it to the corresponding location on LDEV 1, causing a clobbered directory, system code etc. Many times human beings are the cause of integrity problems. A good example of this is stopping the system while it is in the process of coming up, for whatever reason. This can, and usually does, result in a system which will not boot at all.

In any case, loss of integrity results when the system cannot be started. In other words, INITIAL will not complete the startup procedure, and the user is left with a system that will not come up, with all his data on disc and apparently inaccessible. Or is it? This paper will attempt to describe how to recover the system, and the data.

In the following pages, we will discuss the following topics:

1. How to prepare for a system disaster.
2. Reasons for loss of system integrity.
3. What to do to recover data.
4. Description of utilities to help prevent system problems, and recover the system should this become necessary.

It should be remembered that many of the suggestions presented in the following pages are the "ideal case," and are not absolutely necessary for a well managed system. Many users, for reasons of economy or time, cannot follow all of these suggestions to the letter, and do quite well with their systems.

PREPARATION FOR DISASTER

The following are suggestions for what to do before the system gets into a state where it is unusable. While some of them may seem quite obvious, it is surprising how the basics are often times overlooked. As with almost anything, "be prepared" is the rule of the day if one is anticipating a bad situation.

In all but a few cases, a system which will not start probably has corrupt system files and data. In a few

specific instances (covered later) this can be fixed and the system brought up. However, 95% of the time, a reload is in order. It is important to realize this fact. A reload may, and probably will, be necessary to recover the system. This point cannot be emphasized enough. The only good way to build a corrupt system is not to try to fix what is wrong, but to totally start over and build a new system from scratch. Many people try to avoid this step and find themselves in an even worse situation than before. It is very important that the system manager who finds himself having to recover the system accept the fact that the system will be down for a while.

Doing a reload may not be as bad as it sounds. There are five options on a reload: SPREAD, RESTORE, COMPACT, ACCOUNTS, AND NULL. A full description of what these do can be found in section 6 of the System Manager manual. Briefly, though, SPREAD, RESTORE, and COMPACT will attempt to load all files onto the system. Since the typical system manager will be doing the recovery in the midst of angry users ringing the phone off the hook, it is to everyone's best interest to get the system up as soon as possible. Thus, the ACCOUNTS or NULL option, which do not load all files, should be used. A later section will deal with exactly when to use which of these two options.

In order for a reload to be done, however, there must be something to reload from. This brings us to the first and by far the most important preparation to be performed. That is, have a good Sysdump set available at all times. Again, this point cannot be overemphasized. We will define a "good Sysdump" as being one that has an intact and up to date version of MPE on it, a good directory, and the most recent user files on it.

The most important contribution toward this goal is to perform regular Sysdumps. A periodic schedule of dumps is highly recommended, and must be adhered to. A full Sysdump every day would be ideal, but many users simply cannot afford this, in terms of system down time, operator cost and tape cost. The next best solution to this is to do a weekly Sysdump, then partials to the last full on the other days of the week. Generally, most users do the full dumps on Monday, Friday, or on the weekend. This is the most common method, and provides adequate Sysdump coverage at a minimum of downtime and cost.

There is one additional thing the Sysdump provides which many users overlook, and that is the listing that comes out when the dump is over. This listing includes the file dumped, what disc it was on, what the disc address was, and what reel of the dump it was stored on. Appendix A contains a sample output. As we will see later, this document is essential to being able to recover data. Many times, people do not generate the list because it is too bulky, etc. It is worth the trouble! It's one of those things you will desperately need when you can't get it. It is handy, too, for finding which reel of the dump a particular file is on when it is necessary to restore one during normal system operation. Store all listings for any Sysdump set that is currently valid. It is a good idea to keep the full dump listings together with other full listings, partial listings with partials.

It is very important to have the Sysdump tapes, as well as the Sysdump listings, stored in a location where they are readily accessable to the person recovering the system. Many users, just for safety, store the previous full set offsite. This is a good idea, since a fire could wipe out all hardware and tapes. An offsite copy would insure that some kind of system could be rebuilt, even if it were a week old.

The Sysdump is of little use, however, if the the data on it is unreadable. Use newer tapes if possible for the backups. At least keep the tapes cleaned regularly, and don't use a tape more than a few times before being cycled out of the Sysdump sets. This, of course, varies with how often the tape is used and the quality of the tape. There are a few programs in existence to verify that a Sysdump tape set is good, which will be described in detail later under UTILITIES. These, at best, only give an idea that the tape is good, since only the file labels are really checked for integrity. But at least parity errors will usually be detected.

In addition to the Sysdump tape sets, the system cold load tape should be kept onsite at all times. This tape is a special form of the Sysdump tape, usually containing only MPE, system and subsystem files (@.PUB.SYS). This tape is generally made by the account SE at the time a new version of MPE is loaded onto the system. It is advisable to keep the cold load tape for the current version of MPE, as well as the previous version, just in case it becomes necessary to go back one release. The reason this tape is valuable is so that if problems arise, a known, "good" version of MPE can be loaded onto the system. This can be done with the UPDATE option. Most users keep an additional cold load tape, this one reflecting all configuration changes. This is a good idea, although the UPDATE from the HP made cold load tape will not affect the configuration. If this additional tape is desirable, an UPDATE from the HP made tape is advisable before the configuration changes are made. This is to prevent any "glitches" in the version of MPE on disc to propogate through the cold load tape(s). Along the same line, an update before each full sysdump is also a good idea, for the same reasons. This has an additional benifit, that being that regular cold loads are performed. My Customer Engineer friends tell me that the customers that do a cold load regularly have fewer problems than users who don't.

An absolutely necessary tape set to have is the diagnostic tape set. For Series II/III, this consists of two tapes, a CPU diagnostic tape, and a Non-CPU diagnostic tape. On the HP-IB machines (Series 30/33/44/64), one tape or floppy is used, called the Diagnostic Utility system (DUS). The DUS contains both CPU and peripheral diagnostics, including SADUTIL and SLEUTH, combined. CPU diagnostics are used almost

exclusively by the CE's, while the Non-CPU diagnostics can be utilized by users. Indeed, this paper deals primarily with that very subject. Again, this diagnostic tape or tapes must be created when the system is operable. Waiting till "later" to make the tape could spell trouble. Whenever a new version of MPE is installed, a new set of tapes (or DUS) should be created. This is because the diagnostics are updated along with MPE and the subsystems.

Finally, it is a good idea to keep some sort of list of accounts, groups, etc that the system currently has. A :REPORT listing would serve very nicely. While this would not be necessary to recover the system, it will serve as reference from which to decide which accounts to reload first. This would be the case if production accounts were to be brought back online before development. Some suggestions as to what to put on this list would be:

1. Prioritization of accounts, groups etc that are to be recovered, so that the most critical can be brought back first.
2. A list of all critical files that might have to be recovered. This should be a list of MPE file names, so databases should be listed as DB01, DB02, etc.

REASONS FOR LOSS OF INTEGRITY

We have seen the types of things that are necessary to prepare for a system integrity loss. We will now discuss exactly what causes the system to become inoperable, and what we can do to bring the system back, and recover all the data.

As has been said before, the time when danger of system integrity loss is highest is when the system is down and will not come up. While it is possible to have a running system and have most of the data corrupt, at least MPE is still running, and we have the aid of it and a host of utilities, plus the ability to restore older versions of files, etc. Most sites do not routinely bring the system down at night. Instead, they leave all hardware powered on and MPE running. The question is really one of how the system gets to the down state. The most common cause here is the system interruption, or more specifically, the system failure, system hang, and system halt.

A system failure occures when some part of the MPE system calls a procedure called SUDDENDEATH. An integer number is passed to SUDDENDEATH, which is printed in a system failure message along with the current hardware status and return address of the calling entity. These calls are placed in the code by the MPE lab purposely, and are used when an "impossible" situation is encountered, and MPE cannot continue running. The system hang can be caused for an infinite number of reasons. It usually ends up being caused by a hardware resource which ceases to function. Eventually, every user on the system asks for that resource, causing everyone to wait. A slightly different variation is when a hardware device ties up a system table, and everyone suspends when they try to access that table. A silent halt is similar to a system hang, except that the hardware is in a state in which it cannot run. During a system hang, the hardware will run, but since everyone is suspended, it is never asked to. Silent halts are generally caused by bad hardware, although there are a few software problems that can cause them.

The standard way to recover from any of the above three system problems is to:

1. Take a memory dump.
2. WARMSTART.
3. Print off all spoolfiles.
4. Load system with UPDATE option from a good cold load tape.
5. Log failure and recovery action.

Step four is one reason why it is a good idea to keep a good cold load tape available. The cold load will get the system back to a known good copy of MPE, which may remove the source of the problem. If the failure then re-occurs, a more serious problem is indicated, and the local PICS center should be consulted. Step 5 is often overlooked in the haste of the moment. It is important to keep track of what kind of failure occured, and what actions were taken in case the problem escalates in severity. The Gold book supplied with the system is a good place to log these facts, and places are provided under "Historical Records." This becomes very valuable to the SE/CE who must try to piece together a history of the system's problems in order to locate any trends. This history is absolutely essential to correcting certain very sticky system problems.

One thing to be noted here is that the failure (hang, halt) is not necessarily the cause for the system integrity loss. In other words, the failure itself does not go out and cause data to be corrupted. What the failure can do is indicate the source of the problem. For instance, something might have, at some point in time, caused the file label of a system file to be destroyed. When the system tried to access that system file, a system failure occured. If the file is a critical one, such as an IO driver, it is probable that the system would not come up. The failure did not cause the system file corruption, but indicated that the file was, indeed, corrupt. In the process, the system got into a state in which it could not or would not start. A variation of this is when the system interruption occurs during the updating of a critical resource. The net result is the same: a corrupt system. The system is now down and will not come up. In the above example, a cold load may have solved the problem. But let us assume it would not. We now have a system which has a corrupt operating system, and will not come up unless the system is rebuilt from scratch. In other words, we have to do a reload. Thus, system failures can lead to the situtation in which user data must be recovered.

WHAT DO WE DO NOW?

Before discussing how we would save the files, a couple of special "system won't come up" cases should be discussed. One is the situation in which the system was stopped in the middle of a startup. This usually happens when the person doing the startup is in a hurry, and in his haste aborts the startup before it has completed. When another start is attempted, INITIAL displays a message saying something to the effect of: "ALL VOLUMES NOT PRESENT. MOUNT CORRECT VOLUMES OR RELOAD," or "VOLUME TABLE DESTROYED — MUST RELOAD." The reason for this is that in the startup process, INITIAL does several things to insure that the system is in a startable state. It updates the Cold Load ID, which is a number that is stored in the system and changed every time the system is started. The Cold Load ID is kept in many places, and only at the end of the startup procedure can we be sure that all places have been updated with the new number. One of the locations the Cold Load ID is kept is the Volume Table, which has a listing of all disc volume names. If the Cold Load ID in the Volume Table does not match the Cold Load ID kept in the disc volume label, (sector 0 of the disc), then INITIAL assumes something is awry, and will not let the system come up.

In most cases, the system can be brought up safely at this point. The problem is to get INITIAL to ignore the Cold Load ID's. This is done by zeroing out the cold load id's, using the HP utility SADUTIL, which will be discussed in detail a little later. Below is a list of the locations of the Cold Load ID that INITIAL checks, plus a few other things that should be set on disc.

1. Word 7 of Sector 0 of every system domain disc.
2. Word %12 of the Disc Cold Load Table, located on LDEV 1.
3. Word 1 of the Volume Table (on disc). The Volume Table is pointed to by words %124-125 of the Cold Load Information Table.
4. Word %32 of the Disc Cold Load Table contains bits which tell what the previous load was. While not absolutely necessary, this should be zeroed out.

A full summary of this procedure is in Appendix B of this paper. It should be noted that this procedure will not always work, and the locations of the data on disc can be changed by the MPE lab at any time. This is, at best, a kludge which can get a system up and running in very few cases.

Another special case is that the directory itself is corrupted in such a way as it needs to be rebuilt. In this case, it is necessary to build an empty directory (which the ACCOUNTS option does not do) and then rebuild the accounting structure. The easiest way is to use a utility called BULDACCT. This creates two jobstreams. The first builds all the accounts, and the second logs on as the manager of all those accounts and builds users, groups, private volumes, etc. The full sequence of events for this would be:

1. Do a full Sysdump. This will have a corrupt directory on the tape, but that doesn't matter.
2. Log on as MANAGER.SYS,PUB and run BULDACCT. This creates two files, JOBACCT and JOBACCTB.
3. :STORE these two files on a separate tape.
4. RELOAD from the full Sysdump, using the NULL option. This builds a system with PUB.SYS, MANAGER.SYS and only system files.
5. :RESTORE the two files off the tape.
6. :STREAM JOBACCT. This will build the accounts, then stream JOBACCTB, which builds the rest of the account structure.
7. :RESTORE @.@.@ from the full sysdump.

Again, this is only a special case. Usually, the system can't be patched together like this, and a reload is in order. It should be noted that BULDACCT is a user written, unsupported utility.

What happens when, after all you try to do, the system still won't come up? How can data be recovered? The main thing that can be done at this point is to reload the system using an ACCOUNTS option, then restore the most critical user files first, and get the most important applications up and running. To do this, use the list of accounts, files, and users that was discussed in Preparation for Disaster. The critical issue is how to recover any data that may have been updated since the last backup. In some cases, there may have been no updating of files, then there is no problem. Another case may be that the transactions lost may be easy enough to recreate that recovery with SADUTIL is not warranted. In either case, bring the system up as quickly as possible.

At this point, data recovery of the system is critical. To take the data off of disc and store it to tape, we use a utility called SADUTIL. This is a standalone utility which is on either the non-CPU diagnostic tape (Series II/III) or the Diagnostic Utility System Tape/Floppy (HP-IB machines). SADUTIL is written so that all the important functions of MPE, such as the ability to talk to IO devices, read and write to disc, and interpret commands are all contained in one program. Indeed, SADUTIL is essentially a small MPE. In addition, it must fit in Bank 0 of memory. For this reason, SADUTIL does have some limitations, which will be discussed later.

SADUTIL has many functions, but the primary ones that we are concerned with are the SAVE function, which takes files off of disc and writes them to tape; the PDSK function, which prints areas of disc; and the EDIT function, which allows disc locations to be modified. SADUTIL, as well as several other very handy MPE utilities, is fully documented in the MPE UTILITIES manual.

We will assume that the diagnostic tape/DUS is made, (which won't be possible if the system is down) the first step is to load SADUTIL. On Series II/III, this is done by the front panel. On the HP-IB machines, the LOAD button is pressed, and the DUS is loaded into memory. SADUTIL is then selected, and is run by the diagnostic loader. SADUTIL then asks if any configuration changes are to be made. All discs, including floppy drives, must be configured at this point. SADUTIL does not look at the MPE configuration files, but rather has its own internal configuration array. This array can be changed later by using the CONF command. After all configuration changes have been made, SADUTIL prompts the user for a command. This could be a SAVE, PDSK, EDIT or any other SADUTIL command. These commands are listed in the MPE Utilities manual SADUTIL section. Some commands require additional dialog, while others do not.

To save files, enter the SAVE command, and SADUTIL prompts for either the file name, or the disc address. Notice that using the file name assumes the directory is intact. It is better to use the disc ldev and address, which can be obtained by the Sysdump listing. If the file is one that has been created since the last Sysdump, then it will be necessary to use the file name, or use the PFIL command to obtain the disc address. Both of these options assume the directory is intact.

One of SADUTIL's limitations should be mentioned, as it can affect the way recovery can be done. First, SADUTIL cannot handle tape switching. This means that if a file set is given which will span more than one reel of tape, the recovery will terminate. The list of important files to recover mentioned in the Preparation section should include a filesize for each file listed. This is so the proper amount of tape can be estimated. It is very important to back large files one at a time, putting them on a separate tape. It is possible to enter the names one at a time to the SAVE command, and only terminate the list when the end of the tape is near. If a file does spill over, then SADUTIL must be restarted. The file will not be damaged on disc, but the copy on tape cannot be used. Therefore, that file must be saved again on another tape. Be sure and keep a written log of what files are saved in what order on what tapes. This is useful later when these tapes are used to restore the files.

After all files have been saved by SADUTIL, the system must be reloaded. As a rule, the ACCOUNTS option is the fastest way to reload and get applications back online. Before starting the reload, there is one thing that should be done to insure that a complete reload is performed. INITIAL will not reload all of MPE (ie, it assumes that some MPE on disc is valid) if the disc volume label is good. Therefore, it is a good idea to force initial to bring all of MPE off of tape by destroying the volume labels on the system volumes. This is done by using SADUTIL's EDIT command, or use SLEUTH(Series II/III) or SLEUTHSM(HP-IB machines). Appendix C has a sample dialog of how this is done with SLEUTHSM. SLEUTHSM is documented in the Diagnostic Manual Set.

After the ACCOUNTS reload is done, the files are :RESTORED back to disc, partial tapes first, then the full tapes. The full tape should be restored with the KEEP option on the :RESTORE, to insure that files do not get written over by older versions of the same files. This is where the prioritized list of files, accounts, groups, etc, comes in handy. Restore the files in order of prioritization, and this will guarantee the shortest time to applications being back online. This list may be deviated from, since how critical an application is can vary drastically. An accounts payable application will not be as critical if bills were payed the day before as it might be otherwise. After the restore(s) are done, the files must be restored from the tape(s) created by SADUTIL. This is done using the utility RECOVER2. RECOVER2 will prompt for file sets and names, and give the option of keeping files already on disc. Always overwrite the version on disc, since the file on the SADUTIL tape will always be more current than the version on the last partial. Appendix D has a full SADUTIL dialog, showing how to list and save files, then how to run RECOVER2 when the system is up.

UTILITIES

As we have seen, in order to recover files off of disc and perform other functions, proper software tools must be used. We have discussed two of these, SLEUTH and SADUTIL. These, however, are of use only when the system is being recovered. It should be emphasized that prevention is more important than cure, and that all bases must be covered before disaster strikes. The following is a list of the Utilities that exist for prevention, and their function. Some are not HP supported, and should be used with the same caveats as any other unsupported utility, such as SOO, IOSTAT, etc.

SLEUTH — Standalone diagnostic that exercises peripheral devices. Used primarily to format disc packs. HP supported.

SADUTIL — Standalone diagnostic that allows file recovery when system is down. Also allows modification of disc areas, disc condensation, printing of file information (variation of :LISTF). HP supported.

RECOVER2 — Used to restore the tape created by SADUTIL. Used after system is up and running. HP supported.

BADLABEL — Checks validity of disc files. Tells if anything is wrong with a file label, including whether or not the extents point to free space, or to the extents of another file. Used as preventative measure. User written, privileged.

VALIDATE — Checks sysdump tape to see if file labels are valid. Checks to see if parity of directory or MPE portion of tape is good. Also prints

out creator data, etc. Similar to old utility, STAN. User written.

BADFILE — Used to tell what the last file on a Sysdump tape is, if Sysdump aborts. User written.

FLUTIL3 — Used to display and modify any portion of a file label on disc. Used also to purge any bad files. User written, privileged.

BULDACCT — Used to rebuild accounting structure of system. Will not always work if directory is corrupt. User written.

GETFILE2 — Used to restore files off of :STORE and Sysdump tapes if creator does not exist on the system. If run with PARM=1, a SADUTIL tape can be restored. User written, privileged.

DISKED2 — Utility which performs the SADUTIL EDIT function. Allows online modification of disc locations. HP supported.

CONCLUSION

In summary, there are several steps which lead to maintaining system integrity, and to recover it if lost:

1. Do consistent, regular backups. Validate tapes to insure that they are readable to the system.
2. Maintain a library of all documents and software necessary to recover the system. This includes full listings from Sysdumps, current diagnostic tapes, and listings of critical application information (file names, etc.)
3. Keep accurate records of all system interruptions, and what action was taken.
4. If the system won't come up, use SADUTIL and SLEUTH to recover files.
5. Reload system using ACCOUNTS option, then restore critical files first. Use RECOVER2 to restore files saved with SADUTIL.

Following these suggestions will, along with some common sense, provide the necessary procedures to insure that the HP3000 provides quality service to the users. My sincere wish is that no one will ever have to use the information in this paper.

APPENDIX A

The following is a sample output listing of Sysdump. This shows the file dumped, where it is located on disc, and what reel of the dump it was stored on.

```
FILE      .GROUP    .ACCOUNT    LDN   ADDRESS   VOLUME

BADLABEL.PUB        .GOERTZ     1     %73067      1
BANNER  .PUB        .GOERTZ     1     %73143      1
COPYLIB .PUB        .GOERTZ     1     %73400      1
COPYLIBK.PUB        .GOERTZ     1     %75754      1
CRASH   .PUB        .GOERTZ     1     %73340      1
CRASH2  .PUB        .GOERTZ     1     %73351      1
CRASH2P .PUB        .GOERTZ     1     %77257      1
CRASHP  .PUB        .GOERTZ     1     %77324      1
CRASHU  .PUB        .GOERTZ     1     %146011     1
DBBUFFER.PUB        .GOERTZ     1     %77371      1
DBWIZARD.PUB        .GOERTZ     1     %77407      1
DECOM3  .PUB        .GOERTZ     1     %146322     1
DECOMP6 .PUB        .GOERTZ     1     %146453     1
DIRMATCH.PUB        .GOERTZ     1     %73223      1
DISCADDR.PUB        .GOERTZ     1     %77507      1
DSCAN   .PUB        .GOERTZ     1     %77522      1
DSCANTST.PUB        .GOERTZ     1     %146604     1
DUANE   .PUB        .GOERTZ     1     %146616     1
DWNTST  .PUB        .GOERTZ     1     %147156     1
DWNTSTP .PUB        .GOERTZ     1     %147175     1
ENTRYPNT.PUB        .GOERTZ     1     %147203     1
EVERGRN .PUB        .GOERTZ     1     %147213     1
EXAMPLE .PUB        .GOERTZ     1     %147275     1
EXAMPLEP.PUB        .GOERTZ     1     %147465     1
FLABEL  .PUB        .GOERTZ     1     %147476     1
FLIMIT  .PUB        .GOERTZ     1     %147642     1
FLUTIL3 .PUB        .GOERTZ     1     %147705     1
FORTRAN .PUB        .GOERTZ     1     %147764     1
```

FTNLIST	.PUB	.GOERTZ	1	%227775	1
FTNNEW	.PUB	.GOERTZ	1	%77614	1
FTNUSL	.PUB	.GOERTZ	1	%232341	1
GETSTRNG	.PUB	.GOERTZ	1	%150564	1
GRDSCHMA	.PUB	.GOERTZ	1	%150572	1
GRDTEST	.PUB	.GOERTZ	1	%154731	1
GRDTSTF	.PUB	.GOERTZ	1	%154740	1
ID	.PUB	.GOERTZ	1	%154755	1
IDGGEN	.PUB	.GOERTZ	1	%154766	1
KSAMRBLD	.PUB	.GOERTZ	1	%232652	1
LABJOB	.PUB	.GOERTZ	1	%154777	1
LIBREST	.PUB	.GOERTZ	1	%232714	1
LIMCHNG	.PUB	.GOERTZ	1	%155006	1
LSTALLOC	.PUB	.GOERTZ	1	%162677	1
PICS	.PUB	.GOERTZ	1	%162707	1
PICS2	.PUB	.GOERTZ	1	%162714	1
PICTEST	.PUB	.GOERTZ	1	%162720	1
PICTEST1	.PUB	.GOERTZ	1	%233025	1
PRINTER	.PUB	.GOERTZ	1	%233040	1
QDISPLAY	.PUB	.GOERTZ	1	%233054	1
RTMX	.PUB	.GOERTZ	1	%233140	1
SEGPROG	.PUB	.GOERTZ	1	%234675	1
SEP329	.PUB	.GOERTZ	1	%234710	1
SETCOBOL	.PUB	.GOERTZ	1	%234716	1
SETTDPC	.PUB	.GOERTZ	1	%234723	1
SL	.PUB	.GOERTZ	1	%234731	1
SL2	.PUB	.GOERTZ	1	%235076	1
SLPMAP	.PUB	.GOERTZ	1	%235243	1
SLPMAPP	.PUB	.GOERTZ	1	%235440	1
SLPMAPQ	.PUB	.GOERTZ	1	%235466	1
SM	.PUB	.GOERTZ	1	%235646	1
SOO	.PUB	.GOERTZ	1	%237704	1
SPD4800	.PUB	.GOERTZ	1	%235663	1
SPL	.PUB	.GOERTZ	1	%240347	1
SPL2	.PUB	.GOERTZ	1	%235671	1
SPLLAB	.PUB	.GOERTZ	1	%240073	1
SPLLAB2	.PUB	.GOERTZ	1	%235715	1
SPLLAB3	.PUB	.GOERTZ	1	%241327	1
SPLSTD	.PUB	.GOERTZ	1	%240117	1
SPLXREF	.PUB	.GOERTZ	1	%241542	1
SUSTRACK	.PUB	.GOERTZ	1	%241620	1
SWITCH	.PUB	.GOERTZ	1	%241650	1
TAPELAB	.PUB	.GOERTZ	1	%241664	1
TERMID	.PUB	.GOERTZ	1	%241676	1
TEST	.PUB	.GOERTZ	1	%241704	1
TESTER	.PUB	.GOERTZ	1	%241711	1
TESTFILE	.PUB	.GOERTZ	1	%241717	1
TESTVM	.PUB	.GOERTZ	1	%244257	1
UDC	.PUB	.GOERTZ	1	%275247	1
UDCCPL	.PUB	.GOERTZ	1	%235731	1
UDCUTIL	.PUB	.GOERTZ	1	%244435	1
ULDSET	.PUB	.GOERTZ	1	%275325	1
USERINIT	.PUB	.GOERTZ	1	%275336	1
UT817	.PUB	.GOERTZ	1	%275350	1
XXX	.PUB	.GOERTZ	1	%275417	1
XXXP	.PUB	.GOERTZ	1	%275601	1
XYZP	.PUB	.GOERTZ	1	%275611	1

APPENDIX B

The following is a sample dialog showing how to use SADUTIL to set the cold load id's on disc. This is in the event that a load was aborted. This dialog was done on a Series 44 with one 7925 disc.

```
->start
}
HP32033C.F0.D3
WHICH OPTION <WARMSTART/COOLSTART>? COO
ANY CHANGES?
STACK MARKER TRACE
     1     3552   100035      174    MAINSEG1
     0       13   101037        4    BOOTSTRAP


ERROR #201  VOLUME TABLE DESTROYED - MUST RELOAD

HALT  4
->LOAD
%"%

Diagnostic/Utility System Revision 01.01
Enter Your Program Name (type HELP for program information)
:SADUTIL
Disc Utility A01.03 (C) Hewlett-Packard Co., 1976
LIST LOGICAL DEVICES? Y
 LDEV   DRT   UNIT   TYPE   SUBTYPE
 ----   ---   ----   ----   -------

DISC CONFIGURATION CHANGES? Y
LOGICAL DEVICE? 1
DRT? 89
UNIT? 0
TYPE? 0
SUBTYPE? 9
LOGICAL DEVICE?
LIST LOGICAL DEVICES? Y
 LDEV   DRT   UNIT   TYPE   SUBTYPE
 ----   ---   ----   ----   -------

   1     89     0      0        9
SERIAL DEVICE CHANGES? Y
DRT? 73
UNIT? 0
TYPE? 24
SUBTYPE? 0
ENTER FUNCTION: PDSK 1
 ENTER ADDRESS: 0,1;A
 LDEV=    1, DRT=   89, UNIT=    0, TYPE=    0, SUBTYPE=    9
 SECTOR % 0
   0: SYSTEM DISC ....                100: ................
  10: 3000MH7925U0....                110: ................
  20: ................                120: ................
  30: ................                130: ................
  40: ................                140: ................
  50: ................                150: ................
  60: ................                160: ................
  70: ................                170: ................
```

```
ENTER ADDRESS: 0,1;0
SECTOR % 0
  0: 051531 051524 042515 020104 044523 041440 000011 001007
 10: 031460 030060 046510 033471 031065 052460 000000 000000
 20: 000000 000000 000000 000000 000000 000000 000000 000000
 30: 000000 000000 000000 000000 000000 000000 000000 000000
 40: 000000 000000 000000 000000 000000 000000 000000 000000
 50: 000000 000000 000000 000000 000000 000000 000000 000000
 60: 000000 000000 000000 000000 000000 000000 000000 000000
 70: 000000 000000 000000 000000 000000 000000 000000 000000
100: 000000 000000 000000 000000 000000 000000 000000 000000
110: 000000 000000 000000 000000 000000 000000 000000 000000
120: 000000 000000 000000 000000 000000 000000 000000 000000
130: 000000 000000 000000 000000 000000 000000 000000 000000
140: 000000 000000 000000 000000 000000 000000 000000 000000
150: 000000 000000 000000 000000 000000 000000 000000 000000
160: 000000 000000 000000 000000 000000 000000 000000 000000
170: 000000 000000 000000 000000 000000 000000 000000 000000

ENTER ADDRESS: 28,1;0
SECTOR % 34
  0: 000056 000136 000026 000020 043600 173010 030263 026674
 10: 026674 000011 001007 001171 000000 000136 000000 032600
 20: 000003 000020 000000 013716 013560 024000 000000 000005
 30: 000264 000020 000000 002004 000000 000131 000000 033040
 40: 000000 033120 000000 033320 000400 033530 000400 033540
 50: 000000 013723 000000 013726 000003 000003 000220 036617
 60: 000000 032262 000200 042300 000000 032251 002000 040300
 70: 000000 032252 000200 042500 000000 032276 000400 042700
100: 000000 032277 000007 036610 000000 071750 000060 037037
110: 000000 071675 000170 037117 000000 071701 000055 037307
120: 000000 071713 000106 037554 000000 071741 000170 037364
130: 000000 071727 030263 043600 000000 071770 003544 174200
140: 000000 000004 014100 160100 000000 013731 025744 132100
150: 000000 014012 004120 125700 000000 014142 005060 120600
160: 000000 014163 003734 114600 000000 014210 006264 106300
170: 000000 014230 003544 102500 000000 014262 003024 077400

ENTER ADDRESS: %71741,1;0
SECTOR % 71741
  0: 002016 001010 000004 001007 000000 000000 000000 000000
 10: 000000 000000 000000 000000 000000 000000 046510 033471
 20: 031065 052460 000000 000000 000000 000000 000000 032600
 30: 000000 024000 000410 000000 000000 000000 000000 000000
 40: 000000 000000 000000 000000 000000 000000 000000 000000
 50: 000000 000000 000000 000000 000000 000000 000000 000000
 60: 000000 000000 000000 000000 000000 000000 000000 000000
 70: 000000 000000 000000 000000 000000 000000 000000 000000
100: 000000 000000 000000 000000 000000 000000 000000 000000
110: 000000 000000 000000 000000 000000 000000 000000 000000
120: 000000 000000 000000 000000 000000 000000 000000 000000
130: 000000 000000 000000 000000 000000 000000 000000 000000
140: 000000 000000 000000 000000 000000 000000 000000 000000
150: 000000 000000 000000 000000 000000 000000 000000 000000
160: 000000 000000 000000 000000 000000 000000 000000 000000
170: 000000 000000 000000 000000 000000 000000 000000 000000
```

```
 ENTER ADDRESS:
ENTER FUNCTION: EDIT
 >MODIFY 0,7,1
  SECTOR % 0
    7: 001007:=000000
  WRITTEN
 >MODIFY 28,%12,1
  SECTOR % 34
   12: 001007:=000000
  WRITTEN
 >MODIFY 28,%32,1
  SECTOR % 34
   32: 000000:=000000
  WRITTEN
 >MODIFY %71741,1,1
  SECTOR % 71741
    1: 001010:=000000
  WRITTEN
 >
ENTER FUNCTION: STOP
END OF PROGRAM.
Enter Your Program Name (type HELP for program information)
:
->START
IS IT OK TO ABORT SYSTEM (Y OR N)?Y
)
HP32033C.F0.D3
WHICH OPTION (WARMSTART/COOLSTART)? COO
ANY CHANGES?
**WARNING** DEFAULT VIRTUAL MEMORY SIZES BEING USED

DATE (M/D/Y)?1/3/82
TIME (H:M)?16:36
SUN, JAN  3, 1982,  4:36 PM? (Y/N)Y
LOG FILE NUMBER 634 ON
*WELCOME*
:HELLO OPERATOR.SYS;HIPRI
16:36/12/SP#6/SPOOLED OUT
16:36/#S1/13/LOGON FOR: OPERATOR.SYS,OPERATOR ON LDEV #20
HP3000 / MPE IV C.F0.D3.  SUN, JAN  3, 1982,  4:36 PM
:
```

APPENDIX C

The following is a sample dialog using SLEUTHSM to zero out the volume label on a disc. The dialog for SLEUTH, used on Series II/III machines, is similiar. The method of loading the diagnostic is different, since SLEUTHSM is run as a program under AID, while SLEUTH is a standalone diagnostic.

```
->LOAD
%p%

Diagnostic/Utility System Revision 01.01
Enter Your Program Name (type HELP for program information)
:AID

AID 01.01

>  10 LOAD SLEUTHSM
Program Loaded!!
The Next Available Statement Number is
>5000 DEV 0,11,1,100,0
>5010 DB AA,128,0
>5020 WD 0,AA(0),3,0,0,0
>5030 END
>5040 RUN

End of AID user program
>5040 EXIT
Confirm you want to ERASE the current program(Y or N)?Y


Diagnostic/Utility System Revision 01.01
Enter Your Program Name (type HELP for program information)
:SADUTIL
Disc Utility A01.03 (C) Hewlett-Packard Co., 1976
LIST LOGICAL DEVICES?
DISC CONFIGURATION CHANGES? Y
LOGICAL DEVICE? 1
DRT? 89
UNIT? 0
TYPE? 0
SUBTYPE? 9
LOGICAL DEVICE?
LIST LOGICAL DEVICES? N
SERIAL DEVICE CHANGES? Y
DRT? 73
UNIT? 0
TYPE? 24
SUBTYPE? 0
SYSTEM DISC UNINITIALIZED
```

```
ENTER FUNCTION: PDSK 1
 ENTER ADDRESS: 0,1;A
 LDEV=   1, DRT=  89, UNIT=    0, TYPE=    0, SUBTYPE=    9
 SECTOR % 0
   0: 000000 000000 000000 000000 000000 000000 000000 000000
  10: 000000 000000 000000 000000 000000 000000 000000 000000
  20: 000000 000000 000000 000000 000000 000000 000000 000000
  30: 000000 000000 000000 000000 000000 000000 000000 000000
  40: 000000 000000 000000 000000 000000 000000 000000 000000
  50: 000000 000000 000000 000000 000000 000000 000000 000000
  60: 000000 000000 000000 000000 000000 000000 000000 000000
  70: 000000 000000 000000 000000 000000 000000 000000 000000
 100: 000000 000000 000000 000000 000000 000000 000000 000000
 110: 000000 000000 000000 000000 000000 000000 000000 000000
 120: 000000 000000 000000 000000 000000 000000 000000 000000
 130: 000000 000000 000000 000000 000000 000000 000000 000000
 140: 000000 000000 000000 000000 000000 000000 000000 000000
 150: 000000 000000 000000 000000 000000 000000 000000 000000
 160: 000000 000000 000000 000000 000000 000000 000000 000000
 170: 000000 000000 000000 000000 000000 000000 000000 000000

 ENTER ADDRESS:
ENTER FUNCTION: STOP
END OF PROGRAM.
Enter Your Program Name (type HELP for program information)
:
->LOAD
IS IT OK TO ABORT SYSTEM (Y OR N)?Y
)
HP32033C.F0.D3
WHICH OPTION (COLDSTART/RELOAD/UPDATE)? REL
WHICH OPTION (SPREAD/COMPACT/RESTORE/ACCOUNTS/NULL)? ACC
ANY CHANGES?
NON-SYSTEM VOLUME ON LDEV 1
ADD TO SYSTEM VOLUME SET? Y
ENTER VOLUME NAME? MH7925U0
LOGICAL PACK SIZE IN CYLINDERS = 815.?
SUSPECT TRK  LDEV #1 CYL=23 HEAD=6 (SECTORS %32500-%32577)
DELETE OR REASSIGN? DEL
BANK 0 DEPENDENT MEMORY USED - 26884

DATE (M/D/Y)?1/3/82
TIME (H:M)?16:%57
SUN, JAN  3, 1982,  4:57 PM? (Y/N)Y
LOG FILE NUMBER 633 ON
*WELCOME*
:HELLO OPERATOR.SYS;HIPRI
16:57/12/SP#6/SPOOLED OUT
16:57/#S1/13/LOGON FOR: OPERATOR.SYS,OPERATOR ON LDEV #20
HP3000 / MPE IV C.F0.D3.  SUN, JAN  3, 1982,  4:57 PM
:FILE T;DE=VTAPE!!!
FILE T;DEV=TAPE
:RESTORE *T;@,@.@;OLDDATE
?16:57/#S1/13/I.DEV# FOR "T" ON TAPE (NUM)?
=REPLY 13,7
16:57/9/VOL UNLABELLED   MOUNTED ON LDEV# 7
```

```
FILES RESTORED = 186

FILES NOT RESTORED = 25

   FILE      .GROUP    .ACCOUNT FILESET     REASON

   CATALOG   .PUB      .SYS          1      BUSY
   CONFDATA  .PUB      .SYS          1      BUSY
   DEVREC    .PUB      .SYS          1      BUSY
   HIOLPRT0  .PUB      .SYS          1      BUSY
   HIOMDSC1  .PUB      .SYS          1      BUSY
   HIOTAPE0  .PUB      .SYS          1      BUSY
   HIOTERM0  .PUB      .SYS          1      BUSY
   ININ      .PUB      .SYS          1      BUSY
   INITIAL   .PUB      .SYS          1      BUSY
   LOAD      .PUB      .SYS          1      BUSY
   LOADMAP   .PUB      .SYS          1      BUSY
   LOG       .PUB      .SYS          1      BUSY
   MAKECAT   .PUB      .SYS          1      BUSY
   MEMLOGP   .PUB      .SYS          1      BUSY
   PFAIL     .PUB      .SYS          1      BUSY
   PROGEN    .PUB      .SYS          1      BUSY
   PVPROC    .PUB      .SYS          1      BUSY
   SDFCHECK  .PUB      .SYS          1      BUSY
   SDFCOM    .PUB      .SYS          1      BUSY
   SDFGEN    .PUB      .SYS          1      BUSY
   SDFLOAD   .PUB      .SYS          1      BUSY
   SEGDVR    .PUB      .SYS          1      BUSY
   SEGPROC   .PUB      .SYS          1      BUSY
   SYSDUMP   .PUB      .SYS          1      BUSY
   UCOP      .PUB      .SYS          1      BUSY

:BYE

CPU=45. CONNECT=9. SUN, JAN  3, 1982,  5:06 PM
17:06/#S1/13/LOGOFF

:
=SHUTDOWN


HALT  0
->
```

APPENDIX D

The following is a sample dialog showing how SADUTIL can be used to list and recover disc files. **These are then restored using RECOVER2.**

```
=SHUTDOWN

SESSION ABORTED BY SYSTEM MANAGEMENT
CPU=1. CONNECT=1. SUN, JAN  3, 1982,  4:36 PM
16:36/#S1/13/LOGOFF
16:36/1/ALL JOBS LOGGED OFF
SHUT

HALT 15
->LOAD
*^*

Diagnostic/Utility System Revision 01.01
Enter Your Program Name (type HELP for program information)
:SADUTIL
Disc Utility A01.03 (C) Hewlett-Packard Co., 1976
LIST LOGICAL DEVICES? N
DISC CONFIGURATION CHANGES? Y
LOGICAL DEVICE? 1
DRT? 89
UNIT? 0
TYPE? 0
SUBTYPE? 9
LOGICAL DEVICE?
LIST LOGICAL DEVICES? N
SERIAL DEVICE CHANGES? Y
DRT? 73
UNIT? 0
TYPE? 24
SUBTYPE? 0
DRT  #89, UNIT #0 NOT READY
DRT  #89, UNIT #0 NOT READY
ENTER FUNCTION: PFIL
 ENTER NAME: @.PUB.GOERTZ

 ACCOUNT = GOERTZ    GROUP = PUB

 BADLABEL  BANNER    COPYLIB   COPYLIBK  CRASH     CRASH2
 CRASH2P   CRASHP    CRASHU    DEBUFFER  DRWIZARD  DECOM3
 DECOMP6   DIRMATCH  DISCADDR  DSCAN     DSCANTST  DUANE
 DWNTST    DWNTSTP   ENTRYPNT  EVERGRN   EXAMPLE   EXAMPLEP
 FLABEL    FLIMIT    FLUTIL3   FORTRAN   FTNLIST   FTNNEW
 FTNUSL    GETSTRNG  GRDSCHMA  GRDTEST   GRDTSTP   ID
 IDGGEN    KSAMRBLD  LABJOB    LIBREST   LIMCHNG   LSTALLOC
 PICS      PICS2     PICTEST   PICTEST1  PRINTER   QDISPLAY
 RTMX      SEGPROG   SEP329    SETCOBOL  SETTDPC   SL
 SL2       SLPMAP    SLPMAPP   SLPMAPQ   SM        SOO
 SPD4800   SPL       SPL2      SPLLAB    SPLLAB2   SPLLAB3
 SPLSTD    SPLXREF   SUSTRACK  SWITCH    TAPELAB   TERMID
 TEST      TESTER    TESTFILE  TESTVM    UDC       UDCCPL
 UDCUTIL   ULDSET    USERINIT  UT817     XXX       XXXP
 XYZP
```

```
ENTER NAME: BANNER.PUB.GOERTZ,1

ACCOUNT = GOERTZ    GROUP = PUB

BANNER    1    %73143

ENTER NAME: BANNER.PUB.GOERTZ,2

ACCOUNT = GOERTZ    GROUP = PUB

BANNER      12/ 1/81   12/ 1/81   12/ 3/81
```

```
ENTER NAME:
ENTER FUNCTION: SAVE
READY SERIAL DEVICE FOR WRITE
FILE NAME (OR LDEV#,%SECTOR ADDRESS)? 1,%73067
BADLABEL.PUB      .GOERTZ    - CONTENTS OF LABEL
RETRIEVE THIS FILE (Y/N)? Y
BADLABEL.PUB      .GOERTZ    1    73067
FILE NAME (OR LDEV#,%SECTOR ADDRESS)? BANNER.PUB.GOERTZ
DATE?
BANNER   .PUB     .GOERTZ    1  %73143
FILE NAME (OR LDEV#,%SECTOR ADDRESS)? COPYLIB.PUB.GOERTZ
DATE?
COPYLIB .PUB      .GOERTZ    1  %73400
FILE NAME (OR LDEV#,%SECTOR ADDRESS)? 1,%75754
COPYLIBK.PUB      .GOERTZ    - CONTENTS OF LABEL
RETRIEVE THIS FILE (Y/N)? Y
COPYLIBK.PUB      .GOERTZ    1    75754
FILE NAME (OR LDEV#,%SECTOR ADDRESS)?
ENTER FUNCTION: STOP
END OF PROGRAM.
Enter Your Program Name (type HELP for program information)
:
->SSTART
INVALID - USE HELP

Invalid Command or Input
Enter Your Program Name (type HELP for program information)
:
->START
IS IT OK TO ABORT SYSTEM (Y OR N)?Y
)
HP32033C.F0.D3
WHICH OPTION (WARMSTART/COOLSTART)? COO
ANY CHANGES?

DATE (M/D/Y)?1/3/82
TIME (H:M)?16:43
SUN, JAN  3, 1982,  4:43 PM? (Y/N)Y
LOG FILE NUMBER 635 ON
*WELCOME*
```

```
:HELLO OPERATOR.SYS;HIPRI
16:43/12/SP#6/SPOOLED OUT
16:43/#S1/13/LOGON FOR: OPERATOR.SYS,OPERATOR ON LDEV #20
HP3000 / MPE IV C.F0.D3.  SUN, JAN  3, 1982,  4:43 PM
:HELLO MANAGER.SYS

CPU=1. CONNECT=1. SUN, JAN  3, 1982,  4:43 PM
16:43/#S1/13/LOGOFF

16:43/#S2/14/LOGON FOR: MANAGER.SYS,PUB ON LDEV #20
HP3000 / MPE IV C.F0.D3.  SUN, JAN  3, 1982,  4:43 PM
:RUN RECOVER2.PUB.SYS

RECOVER2 C00.00 (C) HEWLETT-PACKARD CO., 1976
WISH TO KEEP EXISTING COPIES OF FILES (Y/N)?N
?16:43/#S2/15/LDEV# FOR "RECOVTP" ON TAPE (NUM)?
=REPLY 15,7
16:43/9/VOL UNLABELLED   MOUNTED ON LDEV# 7
BADLABEL.PUB      .GOERTZ    1   00100072733
BANNER   .PUB     .GOERTZ    1   00100073010
COPYLIB  .PUB     .GOERTZ    1   00100276217
COPYLIBK.PUB      .GOERTZ    1   00100073071
IS THERE ANOTHER RECOVERY TAPE (Y/N)? N

END OF PROGRAM

:BYE

CPU=9. CONNECT=2. SUN, JAN  3, 1982,  4:44 PM
16:44/#S2/14/LOGOFF
```

System Performance and Optimization Techniques for the HP3000

John E. Hulme
President of Applied Cybernetics, Inc.

INTRODUCTION

The purpose of this paper is to introduce the reader to certain techniques which can improve system performance, throughput, and run-time efficiency on HP3000 computers. These improvements will typically reduce response time substantially and generally increase data processing productivity.

This paper will not simply tell you what to do and what not to do. In many cases there are trade-offs involved and it is more important to understand the principles behind the techniques than the techniques themselves. And because analogies often help us to learn by giving us a new perspective, we will make use of a non-data-processing illustration.

SOME BASIC PRINCIPLES

The first thing to understand is that any given computer can execute a finite number of instructions in a fixed amount of time. When that theoretical limit is reached, no amount of tuning can "squeeze" extra instructions into the computer. For the most part, however, computers do not bog down because we ask them to do too much, but rather because we cause them to trip over themselves in the process of doing it.

This leads to the second important principle: At any moment the computer is either (1) doing productive work; (2) getting ready to do productive work; or (3) waiting on some external action before it can proceed with productive work. As a program is initiated, thereby causing a certain sequence of instructions to be executed, we will call the execution of those instructions "productive work." Whether the "productive work" is really necessary or not, and whether it is efficiently or inefficiently organized, are issues to be addressed later. But a more significant fact of computer life is that usually only a small percentage of the computer's time is spent executing application program instructions.

A CRUDE MODEL

To illustrate these principles, imagine a "library for the blind." The librarian sits behind the desk waiting for a blind person to walk into the library. This is the "waiting period." When the blind person arrives, the "getting ready" period begins. The blind person tells the librarian which book to retrieve and by one method or another the book is retrieved. The librarian now begins the "productive work" phase, reading to the blind person from the selected book. When the reading is completed, the librarian may return the book to the shelf or leave it on the desk. Then a new waiting period begins.

If the library is a busy one, we can imagine that one or more assistants might be hired to transport the books between the librarian's desk and the book shelves. Let's imagine that there is one assistant for each wing of the library. The librarian can do more *productive work* (reading to the patrons), spending less time *getting ready* (still look things up in the card catalog, but now dealing with the assistants instead of transporting books). A new type of waiting is introduced, however: waiting for assistants to bring books back.

In this analogy, the librarian represents the computer's central processing unit (CPU), by which all the productive work is accomplished. Like our imaginary library, the HP3000 has only one CPU. To improve throughput we must maximize the CPU's productive time.

Each patron represents a log-on session or job. The librarian's desk represents the computer's main memory. It is of a limited size, merely a workspace, in comparison to the stacks of book shelves which correspond to the mass storage devices. Finally, each assistant represents an I/O channel transferring data to and from disc, for example.

While illustrating some important concepts, this analogy does not accurately model the run-time environment of the HP3000, or any other computer. How could we refine the model to make it more realistic?

THE MODEL REFINED

At the risk of distorting the human situation, let me suggest four refinements which make our model more nearly resemble the actual computer processes:

1. The "library" should be regarded as a collection of (a) read-only instruction manuals and reference tables (programs and constants) and (b) numerous loose leaf volumes (files) containing sheets of current figures and data (records) which may be periodically replaced, revised, removed, or added to.

2. The 'librarian's" job should be generalized to include any type of service that can be performed on the

basis of preprinted instructions and supplied data.

3. The computer always deals with a *copy* of whatever is stored on the disc, and usually just a few records at a time. So let's imagine that instead of asking a library assistant to fetch a particular book, the librarian will specify a limited number of paragraphs or data sheets and will ask the assistant to bring a photocopy of the desired paragraphs (colored paper for instructions; white paper for data).

4. Because the processing speeds of a computer are so great, our model operates in slow-motion by comparison. Allowing that the librarian can do in one hour what an HP3000 can do in one second (i.e., using the scale of one hour for each second), the assistant could handle 20 to 60 requests per hour, and the equivalent of a 60-word-per-minute typist could enter one character every 12 minutes. A 2400-baud rate would be equivalent to a maximum of 5 characters transmitted per minute, and a 600-line-per-minute printer would correspond to one line every 6 minutes.

SLOW MOTION PERFORMANCE SIMULATION

Visualize this scenario from the patron's point of view (refer to Figure 1): You walk into the library, find an empty cubicle (terminal), and make yourself comfortable. You begin to formulate and transmit your library card number and password (log-on) at the rate of no more than 5 characters per hour. (If it will relieve the agony, you may imagine that you spend the time drawing very large, very elaborate block letters). Depending on the facilities available in the cubicle, you will either transmit each letter as it is formulated or accumulate several characters (maybe even hundreds) and transmit them in a burst. In either case, you transmit each letter separately by ringing a bell, and, when you have the librarian's attention, holding up the card with the letter on it. The librarian records each character of your message on a notepad corresponding to your cubicle, then continues with his other business. Finally you send a character which means "that's the end of what I'm sending you."

The librarian eventually verifies that you are a qualified user of the library and sends you back a standard message which allows you to proceed. This process may require the librarian to send his assistant to the book shelves several times, e.g., to get a procedures manual, index of users, table of passwords, welcome message, etc.

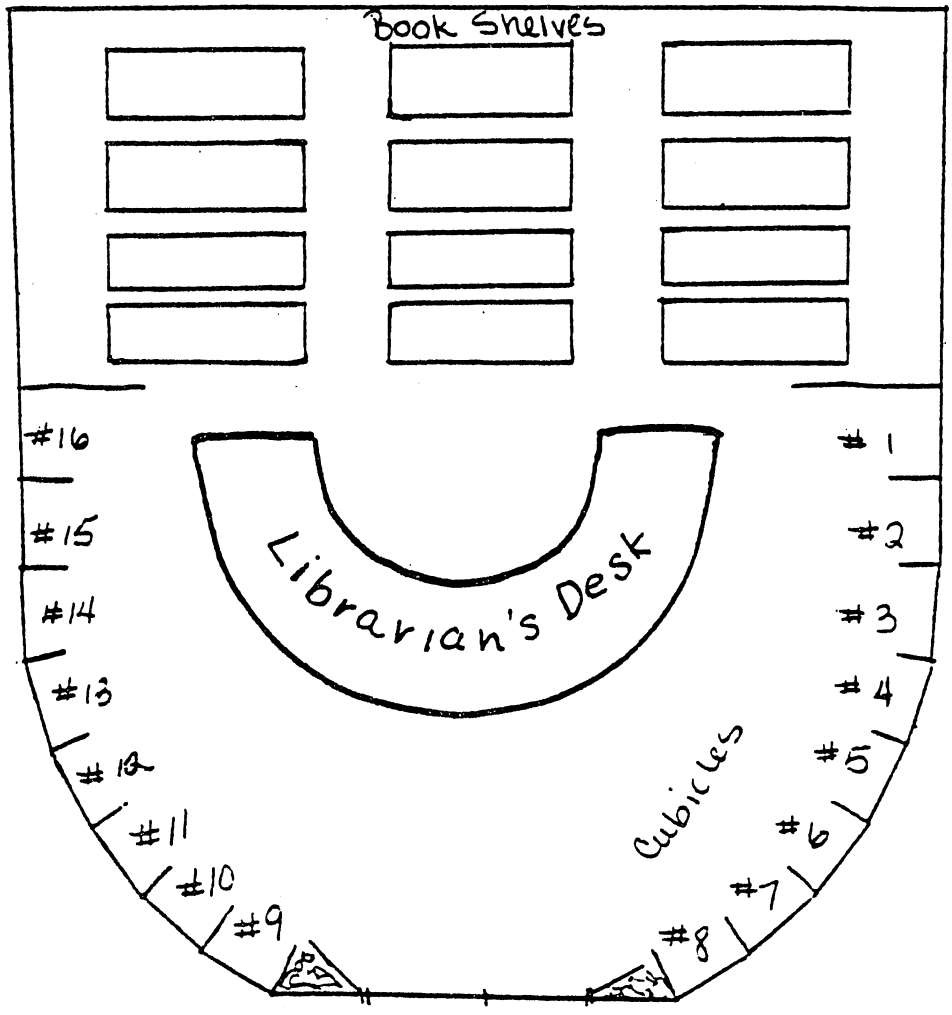


Figure 1. The Library

Next, you painstakingly tell the librarian the name of an instruction manual (program) you want him to follow in performing some service for you. He has the assistant get him a copy of the first paragraph (segment) of the instruction manual (unless a copy happens to be sitting somewhere on the desk already). He also gets a copy, your own personal copy, of a worksheet (your data stack) associated with the specific instruction manual you have specified.

In case there is not enough empty space on the desk for these papers, the librarian first clears some space by either (a) throwing away one of the instruction sheets, (b) having his assistant put the worksheet for some other patron in a special holding file (virtual memory), or (c) having his assistant take one of the data sheets back to the loose-leaf it was copied from and replace the original with the new version.

The librarian now goes to work following the instructions you have requested. This will continue until (a) he comes to a point in the instructions which specifies he is to send certain information to you and/or ask you for additional input; (b) he comes to the end of the page or is otherwise instructed to refer to another page, one which is not currently on the desk; (c) the instructions require that information be fetched from the book shelves, taken there to be filed, or sent to some output device; (d) a predefined length of time elapses (a 500 microsecond quantum corresponds to one-half hour in our model); or (e) the librarian completes his assignment and disposes of your worksheet.

In any of these cases, the librarian will go back to work for one of the other patrons, provided he has all the resources necessary to do so. If not, he will wait (until the necessary information is fetched by the assistant or transmitted by one of the patrons). Depending on what you've asked the librarian to do, and how busy he is doing things for the other patrons, it may take hours or even days before he gets back to you. But then again, it may take days for you to formulate the equivalent of one screen of input, too (at the rate of 5 characters per hour).

THROUGH THE EYES OF THE CPU

Now let's reverse roles and look at the situation from the librarian's perspective. Try to imagine yourself as a calm, unemotional, purely methodical being who is never responsible for mistakes because he does precisely as he is told. You couldn't care less if someone gets poor response time; you aren't to blame, because you only rest when there's nothing for you to do. In fact, you purposely set things aside during peak demand periods to do in your spare time. But you can't take credit for that either — you're only following directions from the MPE handbook.

2:08:17 Ring! There's the bell in cubicle five. He's holding up the letter "R." Write it down on memo pad #5 (line buffer).

2:08:20 Here's the library assistant with the record session #12 requested. Oops! The worksheet for session #12 has been set aside (swapped out to the system disc). Send the assistant for it and wait a minute.

2:08:24 A ring from cubicle #8. That's a carriage return. Time to reinitiate session #8. Make a note to send the assistant for the worksheet when he gets back.

2:08:29 Wait some.

2:09:00 Wait some more.

2:09:16 Oh good, something to do (the observer's feelings, not yours). A ring from cubicle #3. A "7". Write it down.

2:09:20 Here's the assistant. Put worksheet #12 on the desk. Send him back for worksheet #8 — no, there's not room for it. Give him the worksheet for session #5 and send him to file it (we're waiting for input from cubicle #5). We'll send him for worksheet #8 next time.

2:09:24 Okay, now to get to work on task #12. First set the timer for 30 minutes. Now add I to J and put the result in K.

2:09:28 Move W6 to W2. Move . . . hold it, there's another ring from #3. Say, that's only a few seconds . . . must be a block-mode terminal. Write down the "9" and go back to work. Move X to Y. Call the procedure "XFORM." Oh, it's on the desk already — it hardly ever gets thrown out, that's because nearly every program uses it.

2:09:40 Another ring from cubicle #3. This time it's a minus sign. Continue with "XFORM." Convert the first letter of Y to upper-case. Now the second letter. Now the third. Now the fourth. That's all. Return to the main program. It's still in memory. Move the new Y to F3.

2:09:52 Another ring from cubicle #3. A field separator. Resume task #12. Perform FLAG-SET subroutine. It's in another segment, one that's not in memory. Make a note to send for it. Suspend task #12 for a minute.

2:10:04 Cubicle #3 again. Just a blank, but write it down anyway. That's "7-9-minus-field separator-space" so far.

2:10:14 The assistant has finished filing worksheet #5. Send him now for worksheet #8.

2:10:16 Cubicle #3. Another space.

2:10:19 Interrupt from the printer saying the last line has printed successfully. Now reactivate the spooler job — it's instructions are still on the desk and so is the buffer containing the print-line. Initiate I/O transfer.

2:10:26 2-second wait.

2:10:28 Cubicle #3. A third space.
12-second wait.

2:10:40 Cubicle #3. A fourth space. 12-second wait.

2:10:52 Cubicle #3. A fifth space. 12-second wait.

2:11:04 Cubicle #3. A field-separator. 5-second wait.
2;11:09 Worksheet #8 is here. Send assistant to get a copy of FLAG-SET routine. Now to process this input from cubicle #6.
Edit first field. OK. Edit second field. OK. Move first field to R1.
2:11:16 Cubicle #3. The letter "H".
Move second field to K2. Edit third field. Isn't numeric but should be. Transfer to error handler in same segment.
2:11:28 Cubicle #3. The letter "O".
Prepare output to tell cubicle #8 about error. Comment: It's a shame, but since he's in block-mode, he'll have to retransmit the whole screen again, after correcting the error in field 3. And who is to say other errors might not be detected after that? And you, the librarian, can receive those 873 characters, one every 12 seconds for nearly three hours. But you don't mind. It's only a job.
2:11:40 Cubicle #3. The letter "V".
Finish putting error message in the output buffer. Initiate transfer to cubicle #8. Mark task #8 eligible to be swapped out.
2:11:47 Cubicle #11. The letter "P".
2:11:52 Cubicle #3. The letter "E".
FLAG-SET routine is here. Continue with task #12. Move 1 to FLAG. Add 1 to COUNT. Exit back to mainline. What! The assistant had to fetch a separate segment just so we could do that?
2:11:59 Cubicle #11. Oh, oh. Two block-mode devices transmitting at once! Record the letter "I".
2:12:04 Cubicle #3. The letter "R".
Comment Stop, I've had enough of dinging bells! This place sounds like a hotel lobby, not a library!

OBSERVATIONS

As this analogy indicates, there are three factors which reduce overall system performance:

1. Unnecessary disc I/O (most serious);
2. Unnecessary terminal I/O (too common); and
3. Unnecessary CPU usage (rarely the problem in an on-line environment.

Excessive Disc I/O

The primary cause of excessive disc I/O is *inadequate main memory* to hold the required work space (stack and data segments) for each concurrent process, plus all frequently referenced program segments, plus a reasonable mix of infrequently referenced program segments.

The HP3000 is very good at handling multiple concurrent users, even when they won't all fit in memory together. In fact, the use of virtual memory, combined with a well-designed algorithm for selecting which segment to overlay, allows the system to operate efficiently even in cases where a single program exceeds the limits of main memory.

The thing to remember, however, is that code segments put a relatively small load on the system while data segments put a potentially disastrous load on the system. In the first place, code segments can be split up and made as small as the programmer wants them to be. Secondly, they do not have to be rewritten to virtual memory when the main memory space is to be re-used; they are simply overlaid. Data segments, on the other hand, tend to expand, and can be split only with difficulty. Since their contents may change, they must be rewritten each time the process is swapped out, and reread each time it is swapped back in. Finally, whatever data space is required must be repeated for each process that is active. Therefore, if you are supporting 20 terminals, any reduction in data requirements would produce 40 times the benefit that an equivalent reduction in code requirements would produce.

Aside from upgrading to a larger machine, a shortage of main memory can be averted by:

1. Reducing the number of concurrent processes (not an attractive option);
2. Reducing the average stack or data segment size;
3. Reducing the size of the average program segment;
4. Organizing program segments better so that out-of-segment transfers occur less often to non-resident segments and so that often-used code is collected in compact segments that are likely to stay in memory, or
5. Some combination of the above.

When adequate main memory is available, swapping is unnecessary, and disc accesses (which are very expensive in terms of time) will be expended strictly for data retrieval and storage. Once swapping begins, the computer's "productive" activities are at the mercy of "waiting." In the worst case, "threshing" occurs, which means that every time a session gets a turn at execution, either the program segment has been overlaid or the session's work space has been swapped out.

It is worth noting that the use of IMAGE (or of KSAM) causes the allocation of extra data segments. Specifically, each IMAGE database that is open requires a data segment large enough to hold one copy of the root file plus four complete database buffers. If a program accesses multiple databases, or if the root file or buffers are large, the memory requirements will be substantial, and with many terminals running database applications, the memory requirements can add up very quickly. Granted, the advantages of using a powerful access method may outweigh the costs of additional memory demands, but such tools should be used carefully and not indiscriminantly.

It should also be noted that the use of block-mode requires extensive buffers in the stack (at least as large as the largest screen to be transmitted). The use of

VIEW/3000 may add another 6000 bytes of buffer in each user's stack, not to mention the extra data segments created by its use of KSAM. If you have 20 users, this amounts to 120K extra bytes of memory or more.

Excessive Terminal I/O

Major causes of excessive terminal I/O include the following:

1. Transmitting unnecessary characters (trailing spaces, leading zeroes, insignificant digits, etc.) to the computer, a necessary consequence of fixed-format or block-mode input.
2. Transmitting the same data to the computer more than once, as occurs in block-mode when a whole screen is retransmitted to correct an error in a single field.
3. Retransmitting to the computer data which has not been changed since it was received from the computer. This too is the result of block-mode transmission.
4. Repeatedly displaying prompts at the terminal instead of using protected background forms.

Since each character of input consumes critical resources, every effort should be made to ensure that only significant data is transmitted (no extraneous zeroes or spaces and only those fields that are new or have been modified).

It is not only wasteful of computer power, but also destructive of operator morale, to wait until a whole screen of data has been entered and transmitted to the computer before discovering that the screen is invalid due to a duplicate key or an unrecognized search-item value, etc.

It is equally inefficient (for the computer, that is) to display a screen of data, have the operator update a single value and transmit the whole screen back to the computer. In an extreme case, this could amount to over a thousand characters transmitted just to change one or two characters.

Excessive CPU Usage

Besides the costly I/O overhead, it is altogether possible that a retransmitted screen will be completely re-edited, values packed and unpacked, and fields reformatted even though only a single field was updated, and maybe even if *nothing* was updated. This is one cause of unnecessary CPU usage.

Most editing and reformatting done in COBOL subroutines requires excess usage to begin with, and it is far better to allow such work to be done in SPL subroutines, where it can be done efficiently. Including such subroutines in the COBOL programs also causes bulkier segments, which is likely to increase the need for swapping. The best solution is to incorporate all editing within the terminal-handling module itself, since it is already being shared by all on-line programs and is therefore likely to remain constantly in main memory.

There are a multitude of factors which can unnecessarily increase the so-called "productive work" which the CPU has to do. Because computers are seldom CPU-bound in an on-line environment, few people exert the effort to truly optimize CPU performance anymore. Whenever it is a problem, more careful analysis of the program(s) in question will usually yield a more efficient method of solving the application problem.

Often, more careful analysis will also yield a better solution from the point of view of disc I/O as well, both in terms of swapping, code-segment switching, and data retrieval and storage. One word of warning, however: more efficient solutions (CPU-wise) are very often more complex, and to the extent that they increase stack space, or code-segment size, or they require more transfers from one code-segment to another, they may prove counter-productive.

One situation in which heavy CPU usage can be very detrimental is when on-line processes are competing with batch applications for CPU resources. This can be vividly illustrated by running a COBOL compile, an Editor GATHER ALL, a sort, or the BASIC interpreter at the same time on-line programs are running. Block-mode applications exhibit many of these same tendencies and can severely impede response time for character-mode applications when both types are running concurrently.

SPECIFIC OPTIMIZATION TECHNIQUES

1. Resegment programs so that no segment exceeds %5000 words.
2. Set the blockmax parameter on IMAGE schemas as low as possible.
3. Use extra data segments where possible and free them up when finished, rather than increasing stack space for large temporary buffers.
4. Don't keep files open unnecessarily.
5. Don't abuse IMAGE:
   a. eliminate sorted chains where possible.
   b. carefully evaluate tradeoffs of increasing or eliinating secondary paths in detail data sets.
   c. use "@;" or at least "*;" for item lists wherever possible.
   d. only use binary keys (in master file) when overlapping keys can be avoided.
   e. don't let synonym chains get very long.
   f. when loading master data sets, store only primaries on the first pass, makng a second pass for secondaries.
   g. keep master data sets less than 85% filled.
   h. periodically reorganize detail data sets that have long chains associated with a frequently-accessed path (puts consecutive records in the same physical block).
   i. keep the number of data sets in a database as small as practical without requiring many programs to open multiple databases.
   j. keep IMAGE record lengths to a minimum.

6. Have operators exit programs when not in use.
7. Use a field-oriented terminal handler which performs standard edits for you.
8. Use formatted screens with protected background whenever the application is appropriate to such use.
9. Keep terminal I/O buffers small; if possible, eliminate block-mode I/O altogether. (Don't use block-mode and character-mode I/O at the same time.)
10. Don't use VIEW without a lot of memory.
11. Don't use DEL at all.
12. Run CPU-intensive jobs (including compiles, preps, and Editor GATHER ALL) when on-line applications are not running, or at least run them in a lower-priority subqueue.
13. Set the system quantum for a shorter priod than recommended in the MPE manual (but don't overdo it — some experimentation may be necessary).

Auditing with IMAGE Transaction Logging

Robert M. Green
Robelle Consulting Ltd.
Aldergrove, B.C., Canada

SUMMARY

The transaction logging of IMAGE is not just for recovery of lost transactions; the transaction log-files contain a vast array of information that is useful for auditing purposes. Reports generated from these files can answer basic audit inquiries (WHO, WHEN, WHERE, WHAT and HOW), can provide statistics that are useful for performance tuning (which dataset has the most puts and deletes?), and can aid in program debugging (what does this program actually change in the M-CUST dataset?)

CONTENTS

INTRODUCTION

Since MPE release "1918," the IMAGE/3000 database system has had the ability to "log" database changes to a disc or tape file. Although the format of these log-files is somewhat obscure (and not documented accurately or completely), they can provide a great deal of information that is useful for auditing. DBAUDIT (a proprietary software product of Robelle Consulting Ltd.) will analyze transaction logging files and print transaction audit reports from them.

Here are two sample IMAGE transactions (a DBOPEN and a DBPUT), as printed by DBAUDIT, which show the auditing information that is available from transaction logging:

```
OPEN    17 AUG81 11:38 N:3          L:1
U:BOB.GREEN,PUB               P:QDBM.PUB.GREEN
B:TEST.PUB.GREEN            Mode:1   Security:64
Logon device:28   as a session.  Userid:None.
Last dbstore:17 AUG81 11:15

PUT     17 AUG81 11:38 N:4          L:1
U:BOB.GREEN,PUB               P:QDBM.PUB.GREEN
B:TEST.PUB.GREEN            DE:DCOM             R:10
Data-Added:
ORD-NUM          = 11111111
COM-NUM          =       5
COM-DESC         = 55555555555555555555555555555555555555555555
```

Each transaction has a date and time stamp (17 AUG81 11:38), a unique transaction number assigned by MPE (N:3), a unique logging access number for each user who does a DBOPEN (L:1), a logon account, group and user name (U:BOB.GREEN,PUB), a program name (P:QDBM.PUB.GREEN), a base name (B:TEST.PUB.GREEN), a logon device number and batch/session indicator, an optional userid (an extra identifier that can be passed to DBOPEN as part of the password, and acts to distinguish between different users who happen to be logged on with the same MPE user name), the dataset type and name (DE:DCOM is a detail dataset), the data entry's physical record number (R:10), the key-field value (for master dataset entries), the fields that were added, deleted or changed (with field names), plus before and after data values that have been converted from binary to ASCII where necessary (ORD-NUM = 11111111).

Logging Answers Basic Questions

As you can see, logging provides answers to many questions:

```
WHO (logon user name and user id)
WHEN (date and time)
WHERE (database and dataset)
WHAT (data fields changed)
HOW (terminal number and program name)
```

The only question that cannot be answered is WHY?

Two Types of Log-files

There are two basic types of logging files that can contain IMAGE transactions: raw log-files (on disc or tape) which are filled by IMAGE as transactions occur, and user recovery files generated by DBRECOV during transaction recovery.

The original log files have fixed-length records, with large transactions split over several records.

The user recovery files hold the transactions as variable-length records (one record per transaction). User recovery files are usually generated for transactions that DBRECOV could not recovery (and which you must recover by hand). User recovery files contain one extra field which is not found in regular log-files: a recovery flag that indicates whether each transaction was successfully recovered or not ('OK' or 'NO').

SELECTING AND FORMATTING THE TRANSACTIONS

Since a great deal of paper can be consumed in printing every detail of every transaction, DBAUDIT has commands to restrict which transactions are selected and what data is printed for the selected transactions (if any).

The *SELECT* command allows you to select transactions for specified bases, datasets, programs, users, time periods, and a range of record numbers.

```
>SELECT BASE TEST
>SELECT DATASET TEST,DCOM
>SELECT USER BOB.GREEN
>SELECT BEFORE 1132
>SELECT NFROM 212
```

The *LIST* command allows you to control which transactions are printed (*LIST* CALLS) and which field values are printed for the transactions (*LIST* FIELDS). In order to print only the date and time of DBOPEN transactions, these commands would be used:

```
>LIST CALLS NONE
>LIST CALLS O+
>LIST FIELDS NONE
>LIST FIELDS D+T+
```

Here are the full options of *LIST:*

```
>LIST FIELDS N+D+T+C+L+U+P+B+S+R+K+F+M+X+
These flags determine how much information is
printed for each transaction:
L:  unique id number assigned by DBOPEN
U:  user.account,group name
P:  program.group.account
B:  basename.group.account
S:  set name/type; MA=master, DE=detail
R:  IMAGE record number of entry
D:  date of transaction
T:  time of transaction
N:  sequence number assigned by user logging
C:  call type (OPEN, CLOSE, ...)
F:  field values
K:  key-field value
M:  memos from DBMEMO, DBBEGIN, DBEND
X:  extra fields on DBOPEN (mode, etc.)

>LIST CALLS  O+C+P+D+U+B+E+A+M+T+L+
These flags enable/disable listing of the different
IMAGE and MPE intrinsic CALLS in the log-file.
  O=open C=close P=put D=delete U=update B=begin E=end
  A=abort of program between BEGIN and END calls
  T=termination of program without calling DBCLOSE
  L=logging status records (header, trailer, ...)
```

OTHER USEFUL INFORMATION

DBAUDIT also provides three other useful pieces of information for the auditor or database administrator:

1. Reliability of the log-files for recovery purposes.
   DBAUDIT checks each log-file to ensure that transactions are in the proper sequence (for date, time, transaction number and logging access number). If there are any inconsistencies in the log-file, they are detected and reported. Without DBAUDIT, the only way to test a log-file is to restore the original database and actually run a logging recovery. DBAUDIT's double-checking feature has already detected a number of bugs in IMAGE transaction logging.
2. Detection of program aborts.
   DBAUDIT reports program aborts separately from regular program DBCLOSEs. By selecting only the abnormal termination transactions, you can see which programs are aborting. This can be helpful in ensuring program quality.
3. Detection of program "abends."
   DBAUDIT reports programs which terminate with an unmatched DBBEGIN. This can happen because the program aborted during a logical transaction, because the programmer forgot to terminate the logical transaction with a DBEND, or because the system crashed during a logical transaction. DBAUDIT gives you a quick summary count of the number of ABENDs in the file (it should normally be zero), plus additional details if the count is non-zero (where in the file the ABENDs occur, whether the DBBEGIN has a put, delete, or update after it, etc.).

SUMMARY TOTALS AND STATISTICS

From this basic transaction data, it is possible to generate a number of useful summary statistics.

Transaction Breakdowns

One way of analyzing the transactions is to break them down into the different types of transactions, and then total them by program, user, base and dataset:

```
  For all bases, current:    1 entries  20 maximum.
TEST.PUB.GREEN
O:1        X:1        P:2        D:0        U:0
   DCOM               P:2        D:0        U:0

  For all progs, current:    1 entries 200 maximum.
QDBM.PUB.GREEN
O:1        X:1        P:2        D:0        U:0

  For all users, current:    1 entries 200 maximum.
BOB.GREEN,PUB
O:1        X:1        P:2        D:0        U:0
```

In these tables, O equals DBOPENs, X equals DBCLOSEs, DBBEGINs, DBENDs, and DBMEMOs, P equals DBPUTs, D equals DBDELETEs, and U equals DBUPDATEs. Note that for datasets (such as DCOM), only P, D and U totals are collected.

Summary Totals

Here are the types of summary totals provided by DBAUDIT:

```
Logging Records Read from File              32
Transactions that were Selected             6
  Transactions read but not Selected        0
Transactions Selected and Printed           6
  Transactions Selected, not Printed        0
Transactions,but no OPEN (should be 0)      0
Inconsistencies in File (should be 0)       0
Number of ABENDs in file (should be 0)      0
```

FUTURE POSSIBILITIES

Information that could be generated from the log-files, but is not currently collected by DBAUDIT, is the total "changes" to a given numeric field, such as ACCOUNT-BALANCE in a CUSTOMER-MASTER entry. By periodically summing the field values in the entire database and comparing changes in this sum with the incremental changes to that field (as recorded in transaction logging), it should be possible to ensure that all transactions are being logged (i.e., the database is in balance with the transactions).

One problem is the database in which IMAGE schema does not accurately describe the actual data fields. This situation usually happens when users over-

flow the 255 item name limit of IMAGE; it is especially common among users of QUIZ/QUICK (from Quasar Systems Ltd.), IMACS/RAPID (Hewlett-Packard's new tool acquired from David Dummer) and PROTOS (COBOL Generator from Cole and Van Sickle), since these tools use some form of data dictionary to re-define the IMAGE data entries. It is likely that DBAUDIT will be enhanced in the future to provide a user hook (via LOADPROC) to allow non-IMAGE examination and formatting of each transaction.

There is one change to the IMAGE transaction logging that HP could make to improve the audit potential. For DBUPDATE transactions to detail datasets, IMAGE logs only the record number and the data fields that were actually changed. Since the search items and sort items can never be altered by a DBUPDATE, they are not included in the values that are logged. In some applications, this can make it very difficult to determine which record was actually changed (i.e., which customer). For master datasets, the key-field value is always logged. For DBPUTs and DBDELETEs, the critical fields are always logged.

HINTS ON TRANSACTION LOGGING

1. The first thing you need in order to use transaction logging is: answers to a lot of questions. Here is where to look:
   a. PAPERS.QLIBDOC.ROBELLE; this file (which is included on all Robelle product tapes) contains several interesting articles about transaction logging, including information on performance and answers by HP to the most commonly asked questions.
   b. IMAGE MANUAL; this HP manual describes the transaction logging and recovery system, DBRECOV, DBUTIL commands to enable/disable logging and recovery, the DBBEGIN, DBEND and DBMEMO intrinsics, the IMAGE log record format and MPE log record format (do not trust this information completely).
   c. MPE INTRINSICS MANUAL; this manual describes the User Logging Facility (upon which IMAGE logging rests), the OPENLOG, WRITELOG and CLOSELOG intrinsics.
   d. MPE SYSTEMS SUPERVISOR MANUAL; this manual describes the ALTACCT and ALTUSER commands which are needed to give out LG capability (needed by user/account to access the logging facility).
   e. MPE COMMANDS MANUAL; this manual describes the GETLOG, RELLOG, ALTLOG, LISTLOG and SHOWLOGSTATUS commands (these require LG capability) and the User Logging Facility.
   f. MPE CONSOLE OPERATOR MANUAL; this manual describes the LOG command, which can only be executed on the master console and which is needed to start and stop the actual logging process.
2. Should I log to disc or tape?
   a. The log-file may reside on either disc or tape.
   b. If on disc, the integrity of the log-file may be no better than that of your data. If you reach EOF on disc, logging stops, subsequent transactions are rejected, and data may be inconsistent.
   c. If on tape, the tape drive is dedicated to logging for the duration of the logging process. If logging to tape, transactions go to disc temporarily to smooth the data flow. If EOT is reached, transactions go to disc while the operator mounts another reel.
3. Power Fail Recovery (Tape)
   a. After power is back on and the system is running, the console will receive a NOT READY message for the tape.
   b. Press the following buttons on the tape drive in this order:
      LOAD
      RESET
      ONLINE
   c. After the ONLINE button is pressed, the tape will move back to the beginning of data and move forward to recover the log records. The system will then resume logging.
4. Getting Started in Transaction Logging
   a. You will need the cooperation of your system manager, your account manager, and someone at the master console. Arrange this cooperation first, or you will be very frustrated. In this example, it is assumed that you wish to turn logging on for a single database, generate a number of transactions, then disable logging and experiment with printing audit reports via DBAUDIT.
   b. Have the system manager give LG capability to your account, and your account manager give LG capability to your user name.

```
:HELLO MANAGER.SYS
:RUN LISTDIR2.PUB.SYS
>LISTACCT GREEN
  ....
 CAP: AM,AL,GL,ND,SF,IA,BA,PH,DS,MR
```

```
>EXIT
:ALTACCT GREEN;CAP=AM,AL,GL,ND,SF,IA,BA,PH,DS,MR,LG
:BYE

:HELLO MGR.GREEN
:ALTUSER MGR.GREEN;&
:CAP=AM,AL,GL,ND,SF,IA,BA,PH,DS,MR,LG
:ALTUSER BOB.GREEN;&
:CAP=AM,AL,GL,ND,SF,IA,BA,PH,DS,MR,LG
:BYE
```

c. Log on with LG capability, build a log-file on disc, acquire a log identifier with the GETLOG command and link it to the log-file. If you are already logged on, don't forget to log off and log back on (otherwise, you will not have the LG capability that you have given yourself above).

```
:BUILD TESTLOG;DISC=3000,8,1;CODE=LOG
:GETLOG AUDIT;LOG=TESTLOG
```

d. Use DBUTIL to link your database to the log identifier.

```
:RUN DBUTIL.PUB.SYS
>SET TEST LOGID=AUDIT    (assume base=TEST)
>EXIT
```

e. The first three steps are one-time operations; you do not have to do them again (for this combination of user, account, log-file, log identifier, and database). At a later time, you can link other databases to this log identifier. The steps that follow are repeated for each "cycle" of transactions that you wish to capture.

f. Ensure that no one is accessing the database, then use DBUTIL to enable LOGGING for the database. Once you have done that, no one can open the database until the log-identifier has been "activated" at the master console.

```
:RUN DBUTIL.PUB.SYS
>ENABLE TEST FOR LOGGING
>EXIT
:LISTLOG
  LOGID          CREATOR      LOGFILE
  AUDIT          BOB.GREEN    TESTLOG.PUB.GREEN
:SHOWLOGSTATUS
NO LOGGING PROCESS CURRENTLY RUNNING (CIWARN 1230)
:RUN APPLPROG.PUB.GREEN
**** UNABLE TO OPEN DATABASE: TEST ****
LOGGING ENABLED AND NO LOG PROCESS RUNNING
```

g. Now comes the hard part, unless you are located at the computer site. Convince someone to do a :LOG AUDIT;START command at the console. Perhaps a phone call to an influential friend would be useful at this point.

```
:LOG AUDIT;START
```

h. You can tell if logging has been started with the :SHOWLOGSTATUS command.

i. Now log on a number of terminals and generate changes to the TEST database using QUERY, application programs, or SUPRTOOL/Robelle. All of these changes will be logged to the file TESTLOG.PUB.GREEN as they occur.

j. Once again, contact the console and have a command entered to stop the log-file.

```
:LOG AUDIT;STOP
```

k. Terminate the programs that are generating the transactions; when the last database accessor has closed the database, MPE will terminate the log process and close the logfile. Now, use DBUTIL to disable logging to this database (so that DBAUDIT can access it!).

```
:RUN DBUTIL.PUB.SYS
>DISABLE TEST FOR LOGGING
>EXIT
```

l. Run DBAUDIT and specify TESTLOG as the source of input records. If you do not want the report on the lineprinter, you can use a :FILE command to the file DBREPORT to redirect it to $STDLIST or to a disc file.

```
:FILE DBREPORT=$STDLIST;REC=-79
:RUN DBAUDIT.PUB.ROBELLE
>INPUT TESTLOG
>EXIT
```

5. Users report that the system load of transaction logging may not be as bad as was first rumored. Considering the performance improvement that is likely to accompany MPE IV, now may be a good time for users to consider activating transaction logging for their databases.
6. In one of the releases of MPE IV, IMAGE was changed in a way that can affect user programs. Previously, a user program could invoke the DBBEGIN, DBEND or DBMEMO intrinsics whether or not logging was active for the database in question. The intrinsics always returned STATUS=0, as long as the parameters were legal. Now, these three intrinsics return an error (non-zero STATUS) if logging is not active at the time of the call (i.e., the DBBEGIN, DBEND or DBMEMO could not be logged). User programs that don't care whether logging is running, must *not* check the STATUS result. User programs that wish to ensure that the operator has activated logging can now do so by checking the STATUS from these intrinsics.

Transaction Logging and Its Uses

Dennis Heidner
Boeing Aerospace Company
Seattle Washington

For some time database users have been concerned about the integrity of their databases and methods to prevent them from being corrupted. Another concern is performance measurement. When H-P introduced MIT-1918, they also introduced "Transaction Logging." Transaction logging is intended to provide a means of repairing databases which are either damaged or are suspected of being so. There are however many additional benefits to be derived from transaction logging including automatic audit trails, historical records of the database users, and information on the database performance.

The purpose of is paper to discuss the basic concepts of transaction logging, its benefits, and drawbacks. Various logging schemes, such as long logical blocks, and multiple IMAGE databases are discussed. Several different database logging cycles and HP recommended recovery procedures are discussed, and a method of recovering and synchronizing multiple databases is proposed.

Finally this paper covers a user written program which has been used to monitor the database performance, to validate and debug new user-written application software, and provide a complete audit trail for future reference.

INTRODUCTION

Many computers are justified only because they can keep track of large quantities of information in "real time" databases. In such cases it becomes extremely important that the integrity of this information remains consistent.

The database can be destroyed or corrupted in a number of ways. These include program errors, personnel errors, and computer hardware problems. A considerable amount of time and resources can be expended to eliminate most of the program errors, but it is almost impossible to guarantee a perfect program. The second source of inconsistencies is people. While it is possible to protect the information from human error by increasing the complexity of the program or by eliminating the human contact with the machine and its peripherals, both are often undesirable. Finally the third cause is system failures. System failures can be caused by numerous events including such things as fires, earthquakes, vandalism, hardware problems, power failures, and of course, MPE flaws.

We can take steps, however, to protect our investment in the database. There exist several very good programs,[1] such as DBCHECK and DBTEST, which will look for and can correct minor structural problems caused by crashes. But what about the user who must update a critical path in IMAGE? To do so requires a DBDELETE followed by a DBPUT. If the system crashes between the two, there will be no structural damage to be found. If you don't mind losing a $50,000 item or a $100,000 check, you have no worries. . . An effective database protection method is transaction logging. Logging takes many forms, the simplest of which only requires that we file away the paperwork used to generate the modifications to the database. Although this is convenient, it is a poor approach when it comes to recovering the database from a crash or system failure. For instance let's assume that we have a failure after two or three thousand transactions have been entered from terminals at several locations. Who wants to re-enter all the old data, while all the normal work is stacking up?

A better method is to have the computer keep duplicate copies of the information used to make the changes. Then it would only be necessary to instruct the computer to use the duplicate to reconstruct the database following a crash.

There are several ways that computers can be used to generate these duplicate copies. The most efficient method is to write the programs with an intrinsic transaction-logging system. This logging system can either be supplied by HP or could be a custom logging scheme. The problem with custom schemes is that they generally require as much or more design time as many of the applications programs that will use them. Since this work is not readily visible to either the end user or management, there is a temptation to do a quick job. The resultant lack of planning causes poor database and system performance. Additionally, in-house logging schemes only work with the in-house programs. If we use externally-written software (such as QUERY), we may find it difficult or impossible to get these routines to use our logging schemes.

TRANSACTION LOGGING (USERLOGGING)

HP recognized this need for database protection, and developed a version of transaction logging which runs on the HP3000.[2-3] HP's transaction logging is actually a

process which runs under the control of the MPE operating system. If the database is enabled for logging, a logging process then attaches itself to the database when it is opened up for any update access. If the database is opened up in a read-only mode, the logging process is not attached. When the logging process is running it intercepts transactions after the IMAGE check has been made, yet before the actual transaction has been made in the database. This captured data (old and new values) are then blocked up in a buffer in memory. When the memory buffer fills up, the transactions are moved out to a logging file on the disc. If we are logging to the disc only, then this becomes our duplicate copy of the transactions. If we are logging to the tape drive, then the disc buffer is periodically moved out to the tape drive (see figure 1).[4]

If we have a system failure (or any other event which could cause a database inconsistency) then we use a database recovery procedure which uses a good copy of the database and the duplicate copy of the transactions to restore the information in the database to its condition only moments before the crash.

The recovery program which HP supplies is called DBRECOV. The program literally re-works all the transactions in the same sequence as originally made; this repetition assures that the database structure is correct and undamaged.

Once the database has been corrected and brought back into a consistent state, a backup copy is made and a new logging media is used. The act of making a backup copy and using a new logging media is known as beginning the logging cycle.

In order to implement transaction logging, HP introduced several new user-callable DBMS procedures: DBBEGIN, DBEND, DBMEMO, WRITELOG, BEGINLOG, ENDLOG, OPENLOG, and CLOSELOG. These new procedures are extremely useful because they let us define how transactions are logically grouped.[5]

To illustrate the importance of logical grouping of transactions, assume we have two mutually-dependent pieces of information. It is important that if any change is made to one item, the change that is made to the second item must also be made. If either item is not changed, then neither should be modified. We can do this by using DBBEGIN to mark the beginning of the dependent changes, and DBEND to mark the end (see figure 2). The intrinsic routines ensure that if there is a system crash or failure between the DBBEGIN and the DBEND, neither transaction is made. While transaction logging does not guarantee that we will not have crashes, it does provide some relief in recovering from their effects.

Now let's talk about the drawbacks. Anytime we ask the CPU to perform additional work, there is an increase in the overhead cost for our process. The object is to balance the additional workload on the computer with the benefits that we hope to gain.

Every time the memory buffer is moved out to disc, or the disc buffer is moved to magnetic tape, these transfers tie up the disc controller. Although this may be for very short periods of time, one of the biggest problems plaguing many HP3000 sites is slow response time due to a large number of disc accesses.

If we install logging then, our response time may become worse. Your alternative of course is to use absolutely no logging at all! Thus transaction loggings may be one of the necessary evils in life.

LOGGING STRATEGIES

The placement of the calls to DBBEGIN and DBEND can play a crucial role in the success or failure of logging. Since each call to DBBEGIN or DBEND causes a logging record to be written, and thus additional overhead, it is tempting not to use these at all. The people in the logging laboratory at HP wrote DBRECOV to handle both blocked and unblocked transactions (QUERY does not block its transactions). However while this is ideal for existing programs, we may be losing some very valuable information about our databases.

By properly placing the DBBEGIN and DBEND it is possible to measure the performance of our database. This information can later be used to tune-up our applications programs. Additionally proper placement of the calls enhances our crash recovery procedures.

The worst possible thing that we can do is to take the easy way out, calling DBBEGIN when we open up the database and calling DBEND as we close the database. This results in large recovery blocks. As long as we never have a crash everything works fine. However the first time we must recover after a crash, we might find that DBRECOV is unable to help us out. This is because the recovery process tries to resolve all transactions made between periods when the database is inactive. With the long blocking scheme the database is almost always active. DBRECOV will attempt to build a monstrous file to look for dependent transactions, and inevitably fail!

HP recommends that we make all the necessary locks on the database, call DBBEGIN, make the transaction, the call DBEND before unlocking (see Figure 3A). This will ensure that we have a minimum chance of large concurrent blocks.[6]

Another strategy that appears to work well is to call DBBEGIN, then lock the database or sets, and make our updates. Conversely we would unlock, and then call DBEND (see Figure 3B). This method allows us to measure the time between the begin and the end, which reflects the performance of our database. This procedure works quite well, as long as the following conditions are met:

- Always use ASSIGN LOCK OPTION OFF in QUERY
- Our transactions are made by terminals, and designed so that they collect the data from the screen, perform edits, then go through the DBBEGIN, DBLOCK, updates, DBUNLOCK, DBEND.

If you cannot operate under these conditions, then stay with HP's recommendations.

MULTIPLE DATABASES

When HP first introduced transaction logging, they did not make any provisions for synchronizing transactions which span multiple databases. The DBBEGIN and DBEND intrinsics work only for a single database at a time.[7] However with MIT 2028, HP introduced the BEGINLOG and ENDLOG intrinsics. These new intrinsics now make it possible to develop a method for synchronizing multiple database transactions. This is done by calling BEGINLOG before any multiple database transaction, and ENDLOG at the completion of the transaction (see figure 4). A user-written program could then scan the transaction log for complete BEGINLOG-ENDLOG blocks and indentify the record number of the last complete transaction.

To recover the database you then run DBRECOV and specify @@CONTROL EOF=recordnum." It may be necessary to run DBRECOV for each database that was involved.

LOGGING CYCLES

The method and length of our logging cycles depends heavily on the application and previous experience with the computer system's reliability. There were several possible methods proposed by HP during the MPE 1918 update course. These include:

- DBSTORE, then start a new logfile
- DBSTORE, start a log tape, when it fills start a new one, when it fills start another . . ..
- SYSDUMP, start a logfile

The first logging cycle method is the perferred method. It is straightforward, the recovery procedure is easy to follow, and in the event of a system failure, downtime is limited to the time needed to recover one logfile.

The second type of logging cycle should only be used on databases which require backing up, but have very little activity. This is because each logfile complicates the recovery procedure, and adds a considerable amount of time to recover each logfile.

The third logging cycle option omits the DBSTORE. We have found that a DBSTORE takes about 2 minutes for 3 megabytes of database (1600 bpi tape, series 33 computer). At first glance it would appear that the use of DBSTORE wastes time. However DBSTORE sets some internal flags and time stamps which SYSDUMP does not. These internal stamps and flags are used by DBRECOV to provide added protection against using logfiles from the wrong time period.

If you use a SYSDUMP tape, you must remember to request SYSDUMP store all the files. If partial backups are done, the database must be restored from the latest full backup, then restored from each succeeding partial, before DBRECOV is used. Because the time stamp and flags were not set by SYSDUMP, we must then specify that DBRECOV is to ignore all time stamps and flags. This is often difficult or dangerous to do, especially if your system operators are inexperienced.

SYSDUMP should only be used as a backup for the previous two logging methods. If you do not want to have your database stored on your backup tapes, then you should look into Alfredo Rego's STORENOT program. STORENOT allows the creator of a database to "tie it up" so that it is not stored by full or partial backups.

The logfile can reside on either the disc or magnetic tape. It is faster to log to the disc; however, if the reason for the system failure is a disc hardware or free space problem, you could lose both your database and the backup copy of the transactions. The other choice is for the logfile to reside on tape. This has two drawbacks: first, it ties up the tape drive, and second, it periodically requires the CPU to move the logging buffer from the disc to tape. If the system is already heavily loaded this can only worsen the problem.

If you decide to log to a disc file, you should be careful to build the logfile large enough to hold all of your expected transactions plus a reserve. You can obtain a rough estimate of the log size by:

```
# of sectors = 4*number of database opens
             + (number of updates * update rec len)
             + (number of puts * put record length)
             + (number of deletes * delete rec len)
             + 1 for DBEND
             + 1 for DBBEGIN

update rec len (in sectors)
    = (# of items in list
  + update buffer size)/256

delete rec len (in sectors)
    = (# of items in list
  + delete buffer size)/256

put record length (in sectors)
    = (# of items in list
  + put buffer size)/256
```

If the buffer sizes are not known — use the media record size . . . you can get that from a DBSCHEMA compilation.

You can count the # of items in the item list or if "@;" was used then just use the item count in that particular set.

If you are not sure you calculated the size correctly

then use the :SHOWLOGSTATUS command to monitor the number of records in the log. If you run out of space in a disc file while logging, you can put the database in a state similar to a crash; this may require that you go through a complete database recovery procedure!

CRASH RECOVERY

HP implies that a recovery procedure must be followed every time there is a database crash.[8] This can be disastrous. On one occasion we followed the recommended crash recovery procedure, purged the database, restored the database, and started DBRECOV. It bombed, and upon investigation we discovered that approximately 500 transactions had been lost because the logtape was blank due to a tape drive malfunction. Moral of the story: You should first determine the cause of the crash, then verify that the logfile is good via LOGLIST or DBAUDIT.

We also found that it is important to write your applications programs so that they abort to prevent further transactions if they detect a logging problem. It is possible for the program to pass the IMAGE checks for DBDELETES, delete an item, then find out there is a logging problem! The end result is one less item in the database. This becomes especially critical if you are one of the many IMAGE users who have to update critical items by deleting and re-adding.

If the crash is because the logfile was too small and filled up, then the end result of trying to recover is that your data-entry personnel spend hours reconstructing previous transactions. It is better to run a program such as LOGLIST, and find out what data have been effected. Then run DBSTORE, build a new, larger logfile, and start a new logging cycle. One note of caution: we found that parity errors on the tape drive cause a crash whose symptoms are almost identical to those of one caused by running out of space on a disc logfile.

If the crash is because of a system failure, the correct procedure is:

- Perform a memory dump for HP
- WARMSTART (if possible); this causes MPE to try to recover the transactions in the internal disc buffer. (THIS IS VERY IMPORTANT!)
- SHUTDOWN
- COOL or COLDSTART
- Run LOGLIST or DBAUDIT to determine who, what, when and how bad the crash is.
- If the database was not open in an update or modify mode then simply start a new logging cycle and get your users back on.
- If the database was open in an update or modify mode, then purge the database using DBUTIL, restore the database using DBRESTOR and recover using DBRECOV. BE SURE TO START A NEW LOGGING CYCLE!

AUDIT TRAILS

Good data processing applications have some form of built-in controls which allow for the verification of the accuracy of the database. This is especially true if the application is in the banking, inventory control, or government fields. In many applications some form of an electronic "paper trail" is mandatory.

The information which is logged by IMAGE exceeds most audit requirements and can provide the required electronic trail. Transaction logging records information about who, when, where, and how an item or entry was modified. This information can be extracted in several ways. Bob Greene has a package called DBAUDIT which can analyze the log.[9] I have contributed a similar program called LOGLIST (via IUG 1982 swaptape) which can expand the transaction log per directions. It is described in a appendix to this paper.

The audit trail recorded by transaction logging can be enhanced by carefully planned use of the 'text' area on DBBEGIN, DBEND and DBMEMO. We record the information which leads to a transaction when we call DBBEGIN. The results of the update or special error conditions are logged on the DBEND. If needed, DBMEMO is used to record special remarks and initials of the person making the change.

If you foresee a requirement for frequent analysis of the transaction log, it is also important to include a time-stamp as an item in individual data entries. This forces IMAGE to log both the present time-stamp and its previous value. The value of this information is apparent when tracing the history of a specfic data entry. With a time-stamp on your data entries, it is possible to pull and analyze only those logfiles which contain the time interval about the time-stamp of interest. Since analysis of a transaction log takes about 10-15 minutes for 40,000 records, the time saved in this manner can be considerable.

Perhaps more importantly from a programmer's point of view, we can use the audit trail as a method of providing continuous software monitoring. The concensus among data-processing people is that it is virtually impossible to guarantee that a complex program will correctly handle all cases regardless of what data is fed to it. When an error does occur at our site, experience indicates that it is generally several months before we notice that something is wrong. By maintaining transaction logfiles for a sufficient length of time (6 months), it is possible to locate the source of most errors. This makes it much easier to correct latent program errors. In addition we have found that if the problem was caused by human error, the hard-copy printout that can be generated from the log tape goes a long way toward refreshing the memory of the person who made the mistake.

For users at sites whose software must be accepted by Quality Assurance, audit trails have an additional advantage. As part of the acceptance testing on new

releases of our applications programs, we DBSTORE the database, then run the test programs and fully analyze the log. This enables us to provide a visual check on fields and items in a manner easier than using QUERY.

After using the transaction log as an audit trail and debugging aid during the last two years I would estimate that we have saved probably a hundred man-hours which would otherwise have been spent looking for the cause of "freak errors."

As with all good things in life there is a "Catch-22." IMAGE3000 is structured as a closely-knit group of files tied together with the root file. When modifications are made to the database , only the set number, item number and item buffer are logged. If the root file is altered (by using ADAGER, DBGROOM, etc.), then the link between the database and the transaction log is broken. The most obvious problem occurs when the order of data sets is changed with ADAGER's DETSLIDE. Suddenly your Employee-Detail becomes your Part-Master and the log analysis program either bombs or gives ridiculous answers. You have two choices: either don't use ADAGER (not a very realistic choice), or use ADAGER's SCHEMA to generate a dummy version of the database structure as it appeared before changes were made. Then use the editor to shrink the capacity of all the sets down to 3 or 5. Assign this schema some version number and identify on all logfiles under which version of the schema the logfile was made. I have set up a separate group in our account for these "old, shrunk databases." Then when I need to look at an old logfile, I set up a file equation referencing the old "database" and run LOGLIST under that condition.

TRANSACTION-LOGGING PERFORMANCE

There is a great emphasis on designing systems with better response time. For this reason any type of overhead (regardless of how beneficial) is generally shunned. To make matters worse, when HP introduced transaction logging with MIT 1918, they had indicated that there would be a "through-put reduction of 30% for large modication-intensive online applications running 10 or more concurrent processes."[10] Unfortunately the test environment used for that statement was not completely explained. During the past two years we have been using transaction logging on a Series 33 with 768 kbytes and typically 11 active processes. Our experience has shown that there was probably less than 10% reduction in throughput. So, what is the overhead cost of transaction logging?

In order to find out, I wrote a program (DBPERF) which allows me to benchmark IMAGE transactons with and without logging. The benchmarks are deliberately run with as light a load as possible in order to isolate the overhead caused by logging from the effects of other users' activities (see APPENDIX: DBPERF). The results of the tests are shown in Figures 5-7. In Figure 5 we see the comparison of the time to DBPUT verses pathcount, on series 33 and 44 CPU's. As seen in Figure 5 the added overhead caused by transaction logging, is marginal. The anomalies on series 44 data was caused by a user logging on and using FCOPY during the benchmark test. Figure 6 shows the comparison of the time to DBDELETE verses pathcount, on the series 33 and 44 CPU's. The overhead caused by transaction is marginal, again the anomalies on the series 44 data was caused by a user logging on and using FCOPY during the benchmark test. Figure 7 shows comparisons of the time to DBOPEN, DBUPDATE and block transactions with DBBEGIN and DBEND. Earlier I mentioned that logging blocks up the IMAGE transactions (approximately 32 transactions), then moves this buffer out to disc. The overhead caused by this movement is comparable to the roll-in and -out of an inactive user process by the memory manager (MAM).

In most on-line applications the overhead added to the transaction is considerably less than the threshold point at which the system becomes overloaded. However batch jobs are generally another story, if you have batch jobs which require a considerable amount of system resources, run them without logging. Store your database before the job begins, stream the job, and when it completes, then store the database and start a new logging cycle. If you have a crash during the unprotected batch jobs it will only require that you DBRESTOR and rerun the jobs.

PREDICTION OF RESPONSE TIMES

At this point it will be worthwhile to discuss a little queueing theory and how it is used to estimate response times so that we can illustrate the effects of transaction logging on the system. A queue is just a waiting line.[11] When we analyze queueing systems, we talk about such things as number of servers, arrival rate, transaction rate and number of users. The classical example of queues in operation is the waiting lines at banks. With only one cashier (number of servers), if the customers arrive at a rate of one per hour (arrival rate) and the cashier takes only 15 minutes to complete an average transaction (transaction rate), then there will be no waiting line and the cashier can perform some overhead functions such as washing windows while waiting for the next customer. If, on the other hand, customers arrive every 15 minutes, then we can expect to find a person at the cashier constantly. The windows start to collect dirt and grime since the cashier no longer has time to wash them. When the arrival rate of the customers increases to one every 10 minutes, we soon find that a line is forming. If sufficient time is allowed to pass, customers start to switch banks, the cashier demands a raise and the windows now appear to have several layers of dirt and grime and strange creatures crawling on them.

Transaction processing on an HP3000 performs in a

similar manner. As long as the arrival rate is sufficiently slower than the transaction rate, MPE is able to perform its necessary overhead functions and the response time is good. Unfortunately the HP3000 cannot ignore its overhead as the cashier did, so as the arrival rate approaches the transaction rate, response time begins to suffer.

It is possible to estimate the response time of the computer if you are able to estimate the number of users, the average time each user "thinks" about what needs to be done, and the time required to complete the transaction. The average "think time" is equal to:

$$\text{Think time} = \frac{\text{arrival rate}}{\text{number of users}}$$

For example:

The XYZ Company has an HP3000 Series 33 computer on which they wish to implement an application which will support 10 users. The "think time" of these users is about 30 seconds each per transaction. The transactions consist of a DBDELETE and a DBPUT on a detail set with four paths. What will their transaction response time be?

The transaction response time is equal to:

$$\text{Transaction response time} = \text{Queue length} * \text{transaction rate}$$

$$\text{Queue length} = \text{the greater of } 1 \text{ or } \frac{\text{number of users} * \text{transaction rate}}{\text{think time}}$$

Using the IMAGE benchmark results, we then determine:

$$\text{Transaction response time} = \text{Queue length} * 1.3 \text{ sec}$$

$$\text{Queue length} = \frac{10*1.3}{30} = \frac{13}{30}$$; as noted above, use 1

then Transaction response time = 1.3 sec

If XYZ adds logging, it will be:

$$\text{Queue length} = \frac{10*1.4}{30} = \frac{14}{30}$$; as noted above, use 1

then Transaction response time = 1.4 sec

Our model works well as long as the computer has time to perform its overhead functions, i.e. code-segment swapping, MAM function, and garbage collection. The time available for the computer was approximately:

$$\text{Computer idle time} = \frac{\text{User think time}}{\text{number of users}} - \text{transaction rate}$$

In the case of the XYZ company this averaged 1.6 seconds per user transaction (with logging).

The overhead that was added due to transaction logging is:

$$\text{Added overhead} = \frac{\text{transaction time with logging} - \text{transaction time without logging}}{\text{transaction time without logging}} * 100$$

or, for XYZ,

$$\text{Added overhead} = \frac{(1.4-1.3)}{1.3} * 100 = 7.6\%$$

If 7.6% overhead is enough to cause XYZ's machine to have problems, can you imagine what an additional user using QUERY, the editor, or any of the compilers would do?

An additional benefit from transaction logging is that we are able to collect the arrival rates, transaction rates, and number of users during our actual production enviroment. With this knowledge we can make more accurate design decisions when developing new and additional applications.[12]

CRASH-PROOF?

How crash-proof is your database? Damage to databases can be caused in several ways. The typical cause of damage is a crash occurring while adding or deleting an item to or from a detail set. If the DBPUT or DBDELETE was manipulating the internal pointers in the database, then you can probably count on having at least one broken chain. Other types of database crashes occur when MPE or some "neat" privelege-mode program adds its own kind words to a random data set! When discovered, this error has the same symptoms as a broken chain; however, you may also be missing a considerable amount of data.

Perhaps the worst kind of database crash is the one you can't find. That is, DBTEST, DBCHECK, ADAGER and even DBUNLOAD-DBLOAD say everything is ok. These errors occur when the data set has a critical path which must be updated. Since IMAGE will not let us update critical paths, we have to delete and re-add. If a crash occurs after the DBDELETE is complete and before the DBPUT re-adds the item, then we have lost an entry in the database though the database structure remains intact (see Figure 8). DBTEST, DBCHECK and the other routines have no way of testing for or detecting this error. If your HP3000 is an accounting system, this is intolerable. This type of error could be prevented by using transaction logging and placing the DBBEGIN at the start of the transaction and DBEND at its finish.

It is possible to estimate your chances of having some form of damage to your database in the event of a crash. This Crash Figure of Merit (CFOM) is given by:

$$CFOM = \frac{(\text{transaction rate} * \text{number of users})}{\text{think time}} * 100$$

If your CFOM is high, say 20 or 30 percent, then it is probably worth the effort to run DBTEST and DBCHECK on every database that was open when a crash occurred. It may also be very much worthwhile to try transaction logging. If the CFOM is very low (one to two percent), then it is probably easier to manually correct errors and run DBCHECK at some convenient time.

SUMMARY

This paper discusses the merits and drawbacks of transaction logging, and provides some basic guidelines to aid in the successful implementation of transaction logging. Since most applications are designed to "earn" money, it is only fair to treat transaction logging in the same manner. As summarized in figure 9, the decision to log or not to log should be made only after a careful review of the associated system costs, its performance cost, alternatives, and by establishing values for the intangibles such as improved data security, benefits from audit trails, etc.

ACKNOWLEDGEMENT

I wish to thank the HP sales office in Bellevue, Washington, for allowing me to run DBPERF on their Series 44.

APPENDIX LOGLIST

LOGLIST is a logfile analysis program written by the author; it has the following capabilities:

A. Show who, what, when, and how a database which was running with transaction logging was accessed.

B. Trace the changes made to the database and expand the values in a format similar to QUERY so that the dump is easily readable.

C. Selectively track user-requested database items which fall within user-specifiable limits.

D. Show when the log was opened, closed, or restarted and identify all users that were accessing the database during a crash!

E. Provide statistics showing the database activity, transaction elapsed time, detail sets accessed, the ratio of BEGIN-ENDS to database transactions, average transaction times, and worst-case transaction response time.

F. F. Identify (if any) the processes which had "broken" transactions.

Running LOGLIST

LOGLIST should be run in the same account and group in which the database resides. If the log to be examined is on disc, then that file must also be accessible. LOGLIST cannot analyze a logtape that is cur-

rently active. Finally, the log analysis consumes considerable CPU time (even though the elapsed time of the analysis may be very short). It is advisable the log analysis be either streamed in a low JOBPRIORITY (DS or ES) or run during periods of low computer usage.

LOGLIST Commands

LOGLIST commands are listed below, each followed by a short summary of its function.

HELP — print additional instructions

DATABASE=[dbname[.group[.acct]]]
(if not specified the values are set to @.@.@ and no expansion of the log records may be done. Only the Log User Summary and histograms will be generated.)

PROCESS=[program[.group[.acct]]]
(if not specified the values are set to @.@.@)

LOGON=[user[.group[.acct]]]
(if not specified the values are set to @.@.@)

LIST[=range]
expand the transactions made to the database (in the QUERY report format) showing:

the user that made the modification
if an UPDATE, what was changed
if a DELETE, what was deleted
if a PUT, what was added

The transactions are outlined in asterisks (*) to indicate indicate "logical transactions." When the beginning or end of a transaction cannot be determined, the program leaves the outlined block open (see Figure 10). On such blocks, the LOGID of the process is printed and it is possible to rerun the analysis — specifying that those items be expanded separately.

RANGE — The range field is optional, and is in the following form:
LIST=startingrecord:endingrecord
If the ending record is not supplied then LISTLOG will continue to expand until the end of the log file.

NOLIST disable expansion of the transactions made to the database

DATE=m1/d1/y1 [TO m2/d2/y2]
look only for transactions made between and including the specified dates. The default for m2/d2/y2 is 99/99/99.

TIME=H1:M1 [TO H2:M2]
look only for transactions made during the specified time interval. The default for H2:M2 is 24:00.

FIND dset.itemname (EQ,LT,GT[,IB])
'value1'[,'value2']
look only for transactions made to dset.itemname and falling within the bracketed area as specified by the relational operators.

FIND dset record#
look only for transactions made to record# of dset.

```
        { TAPE;LABEL=label          }
LOGFILE={                           }
        {filename[.group[.acct]]    }
```

if a filename is specified, you must have exclusive read-access to the file. If tape is specified, you must be able to use this non-sharable device.

RUN — begin processing the transaction log.

EXIT — exit the program and return to MPE.

SHOW — display current parameters.

INIT — initialize the files, plots and data back to the way they were when LOGLIST first started. Any data accumulated so far will be sent to the LP.

LIMIT — limit and identify the "worst" transactions. This causes all transaction response-time data which exceeds 20 times the current running average to be thrown out. The time of day, user and process are printed on $STDLIST. This command has no effect until ten logical transactions have been completed. It is useful in locating deadlocks.

<CONTROL Y> — ("CNTL" and "Y" keys pressed simultaneously) interrupt the program (sessions only). The program will give the the time and date of the transaction which it is currently processing and ask if you wish to continue. A "Y" or "N" is expected.

Interpretation of the Log User Summary (see Figure 11)

USER — Logon user name

GROUP — Logon user's group

ACCT — Logon user's account

DBASE — Database that was accessed

PROCESS — Process run by user

GROUP — Group in which the process resides

ACCT — Account to which the Process belongs

LOGON TIME — Time the process began

LOGOFF TIME — Time the process closed the database

LG# — LOGID # for the process (assigned by MPE)

DEV — Logical device from which the process was run

O — Database open mode

CAPABILITY — User's capability (see WHO intrinsic of MPE)

UP — Number of DBUPDATES

PUT — Number of DBPUTS

DEL — Number of DBDELETES

#BLKS — Number of complete logical transaction blocks

Inferences from the LOGLIST Statistics

Several histograms and charts are derived from the data; these are provided by LOGLIST to aid in the interpretation of the data.

DATABASE ACTIVITY (see Figure 12)

The DATABASE ACTIVITY histogram plots the number of transactions on the y-axis and the time of day (in 15 minute intervals) on the x-axis. This histogram can be useful in determining when the peak database loads occur.

DATABASE RESPONSE TIME (LOG10) (see Figure 13)

The LOG10 plot is a useful tool in determining if a process or processes are suffering from very bad response time or may be causing database deadlocks. The LOG10 plot covers the range from .1 sec to 10,000 seconds.

DATABASE RESPONSE TIME (LINEAR) (see Figure 14)

The LINEAR plot is a useful in determining if a process or processes are suffering from poor database response times. The y-axis represents the number of transactions made. The x-axis represents the time, from 0 to 30 seconds.

LOGICAL BLOCK SIZE (see Figure 15)

The LOGICAL BLOCK SIZE histogram is useful in evaluating the effectiveness of the transaction blocking of a process. This chart may also be used to determine if a program is calling the DBBEGIN-DBEND pair only at the beginning and end of processes or after making single database modifications.

DATABASE RESPONSE TIME (AVERAGE) (see Figure 16)

The AVERAGE histogram can be useful in evaluating modifications made to existing programs by aiding in the determination of whether or not the system (as seen by the database users) is getting slower or faster.

DATABASE RESPONSE TIME (WORST CASE) (see Figure 17)

The WORST CASE histogram is useful in locating processes that may have caused database deadlocks. The histogram is also useful in determining if there are certain times during the day in which stream jobs may be run with little or no impact on the response time for on-line users.

TRANSACTION FREQUENCY (see Figure 18)

The TRANSACTION FREQUENCY histogram is a measure of the time between logical blocks, often called the user's "think time." This plot, in conjuction with the database response time charts, can be helpful in determining if and/or how improvements can be made to the application programs and the system.

ADD-DELETE-UPDATE TO BEGIN-END RATIO (see Figure 19)

The ratio of DBPUTS, DBDELETES, and DBUPDATES to DBBEGINS and DBENDS is a good indication of how the transactions are blocked by the user's application programs. The desirable range is $0 < [\text{PUTS} + \text{DELETES} + \text{UPDATES}] / [\text{BEGINS} + \text{ENDS}] < 100$.

If the ratio is less than one, this usually indicates that there is a process or processes which are making only one database transaction per BEGIN-END set. Although this is not harmful, it does not fully utilize the benefits of transaction logging, resulting in more overhead during the logging process and during recovery.

AVERAGE + STANDARD DEVIATION

LOGLIST provides the averages for the response time and block lengths. With the averages and the standard deviations which are also supplied, it is possible to determine your chances of attaining desired response times or block lengths. For instance, the interval covered by the sum of the average plus one standard deviation includes approximately 85% of all data base transactions logged.

DETAIL SET (DATA BASE) SUMMARY

The DETAIL SET summary provides totals based on the actual activity in the sets. As shown in Figure 20, this information includes the number of DBDELETES, DBPUTS, and DBUPDATES. The capacity and number of entries are also printed.

How LOGLIST Works

When processes are using the "USER LOGGING" facility of MPE, the process opens up a path to the transaction log for each process and each database enabled for logging. As part of this "opening" procedure the user's name, acct, process name, capability, LDEV, and database (if one) are logged in a special record. LOGLIST looks for these records and builds its internal working tables from them.

As processes make transactions to their databases, the logging process intercepts a copy of the changes, adds a time and date stamp then routes them to the logging file. LOGLIST uses the time stamp from the DBBEGIN and DBEND records to determine the total elapsed transaction time. (If you don't use DBBEGIN or DBENDS then you can never measure your response times with LOGLIST!)

Broken transactions can be located by looking for a special "ABNORMAL END" record, and by checking to make sure that all process issued a DBEND before closing the log and terminating.

If the process did not (or was unable) to close the log before terminating, and LOGLIST detects an EOF on the log then it is assumed that there has been a system crash. System crashes can also be determined by looking for the crash marker which was written out at the time of a WARMSTART recovery.

Transactions are expanded by using the information gathered when the process first opened up the log, and the actual data- base "change" records. (These records are marked with "DE," "PU" or "UP.") LOGLIST uses the item-list recorded as part of the transaction and calls DBINFO to determine the types and lengths of the individual items logged.

APPENDIX DBPERF

This program was written to benchmark the time required to perform a wide range of DBPUTS, DBDELETES and DBUPDATES. The primary area of interest was the overhead added to IMAGE/3000 when the user is using transaction logging.

The benchmark process follows the procedure listed below:

A. Disable the database XYZ for logging

B. Perform 50 DBOPEN's and DBCLOSE's to measure time to initially startup the logging process. (NOTE: this will really clobber the response time for everybody else.)

C. Perform 50 DBPUTS to a detail set which contains a single path and various data types. The data used for these operations is generated using the RAND function from the compiler library.

D. Perform 50 DBDELETES to the detail set.

E. Setup a loop so that we can perform 50 DBPUTS and DBDELETES on detail sets which contain from 0 to 15 paths.

F. Generate the plots and data summaries.

G. G) Enable database XYZ for logging, then repeat steps B) thru F)

The database modifications are performed without signaling the start of the transactions with DBBEGIN or the end with DBEND. This was done so that the comparison could be made, without the overhead added by the BEGIN-END blocking. This type of test is fair since the DBBEGIN and DBEND calls are made only to signify that that are changes which are dependent.

The time required to perform the BEGIN-END block is measured and plotted on a separate chart. It should be noted that since DBBEGIN and DBEND do not require immediate access to the disc drives, the time required to perform these intrinsics is very low. The can however add a significant number of records to the memory buffer, which of course means that there is an additionaly load on the I/O channel which controls the disc drives.

REFERENCES

[1]F. Alfredo Rego, "DATABASE THERAPY: A practitioner's experiences," in HPGSUG 1981 Orlando Florida Proceedings, Vol 1, pp. B12-01 to B12-13

[2]P. Sinclair, "MPE 1918: A BONANZA OF ENHANCEMENTS," in COMMUNICATOR issue 23, pp. 4-17

[3]HP, "MPE III 1918 USER UPDATE COURSE"

[4]HP, "MPE III Intrinsics Reference Manual," pp. 3-92 to 3-96

[5]HP, "IMAGE Data Base Management System reference manual," pp. 4-22 to 4-23

[6]HP, "IMAGE Data Base Management System reference manual," pp. 4-23

[7]P. Sinclair, "MPE 1918: A BONANZA OF ENHANCEMENTS," in COMMUNICATOR issue 23, pp. 14

[8]HP, "MPE III 1918 USER UPDATE COURSE," pp. 60

[9]Robert M. Green, Robelle Consulting Ltd., 5421 10th Avenue, Suite 130, Delta, British Columbia V4M 3T,. Canada.

[10]HP, "MPE III 1918 USER UPDATE COURSE," pp. 71

[11]A. O. Allen, "Queueing Models of Computer Systems," in COMPUTER, pp. 13-24, Apr. 1980 (an IEEE publication)

[12]C. Storla, "MEASURING TRANSACTION RESPONSE TIMES," in 1981 IUG Orlando Florida Proceedings, Vol. 1, pp. C7-01 to C7-08

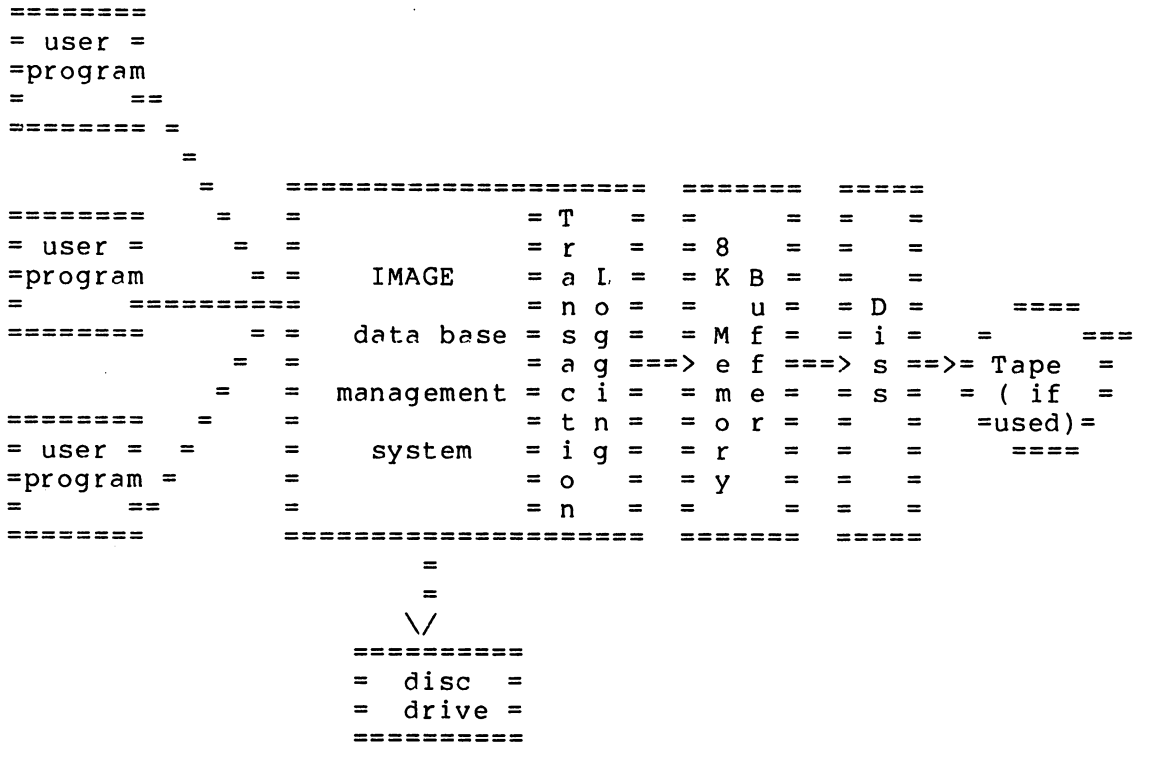


Figure 1. IMAGE transaction logging flow

```
CALL DBBEGIN(BASE,...   )
     CALL DBLOCK(BASE,...  )

     CALL DBDELETE(BASE,... )

               o                          At this point,
               o                     if there is a crash we lose
               o                     this data entry!
               o

      change made to search item

               o
               o
               o

     CALL DBPUT(BASE,... )

               o                 this item has now been re-added

     CALL DBUNLOCK(BASE,... )
CALL DBEND(BASE,...   )
```

Figure 2. Dependent Changes

```
   CALL DBLOCK(... )
        CALL DBFIND(... )
        CALL DBGET(... )

        ** MAKE CHANGES TO ITEM VALUES HERE **

        CALL DBBEGIN(... )

                   DBPUT
             CALL {DBUPDATE } (... )
                   DBDELETE

        CALL DBEND(... )
   CALL DBUNLOCK(... )
```

Figure 3A

```
   CALL DBFIND(... )
   CALL DBGET(... )

   ** MAKE CHANGES TO ITEM VALUES HERE **

        CALL DBBEGIN(... )
        CALL DBLOCK(... )

                  DBPUT
            CALL{DBUPDATE}(... )
                  DBDELETE

        CALL DBUNLOCK(... )
        CALL DBEND(... )
```

Figure 3B

```
CALL BEGINLOG(... )

    CALL DBFIND(BASE1,... )
    CALL DBGET(BASE1,... )

    ** MAKE CHANGES TO ITEM VALUES HERE **

        CALL DBBEGIN(BASE1,... )
        CALL DBLOCK(BASE1,... )

                  DBPUT
             CALL{DBUPDATE}(BASE1,... )
                  DBDELETE

        CALL DBUNLOCK(BASE1,... )
        CALL DBEND(BASE1,... )

    CALL DBFIND(BASE2,... )
    CALL DBGET(BASE2,... )

    ** MAKE CHANGES TO ITEM VALUES HERE **

        CALL DBBEGIN(BASE2,... )
        CALL DBLOCK(BASE2,... )

                  DBPUT
             CALL{DBUPDATE}(BASE2,... )
                  DBDELETE

        CALL DBUNLOCK(BASE2,... )
        CALL DBEND(BASE2,... )

CALL ENDLOG(... )
```

Figure 4

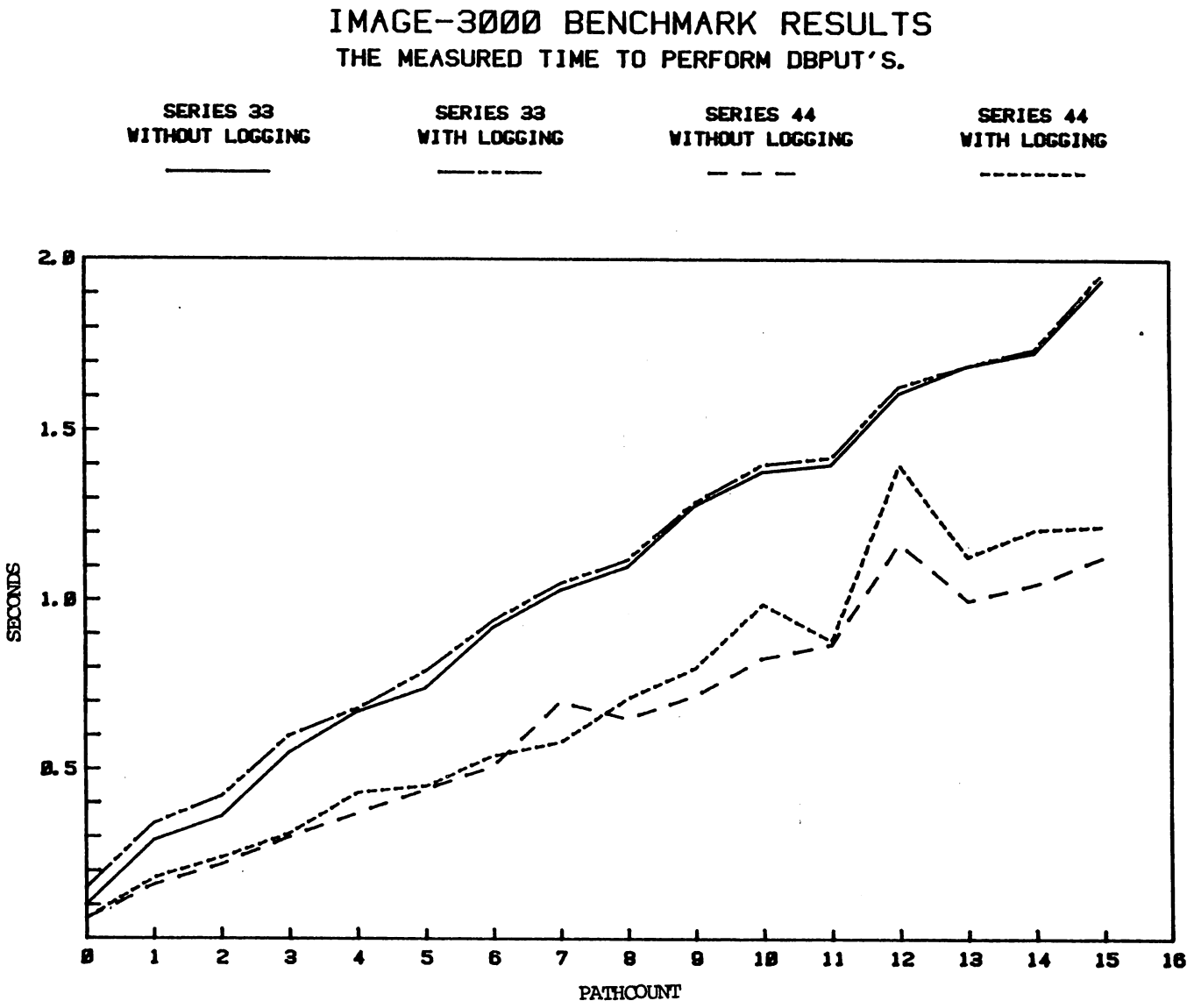


IMAGE-3000 BENCHMARK RESULTS
THE MEASURED TIME TO PERFORM DBPUT'S.
SERIES 33 WITHOUT LOGGING
SERIES 33 WITH LOGGING
SERIES 44 WITHOUT LOGGING
SERIES 44 WITH LOGGING
2.0
1.5
1.0
0.5
SECONDS
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
PATHCOUNT

Figure 5

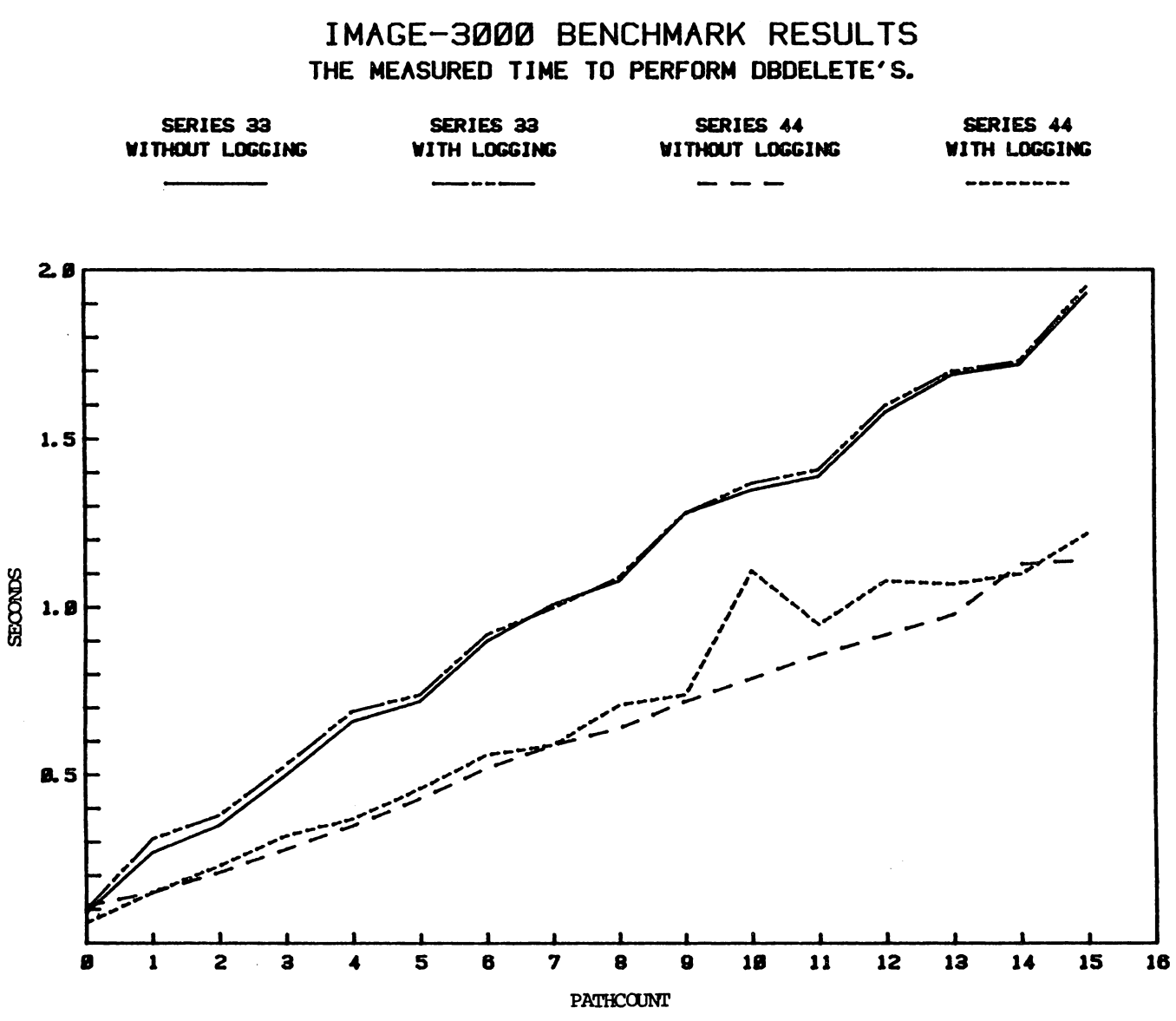


IMAGE-3000 BENCHMARK RESULTS
THE MEASURED TIME TO PERFORM DBDELETE'S.
SERIES 33 WITHOUT LOGGING
SERIES 33 WITH LOGGING
SERIES 44 WITHOUT LOGGING
SERIES 44 WITH LOGGING
2.0
1.5
1.0
0.5
SECONDS
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
PATHCOUNT

Figure 6

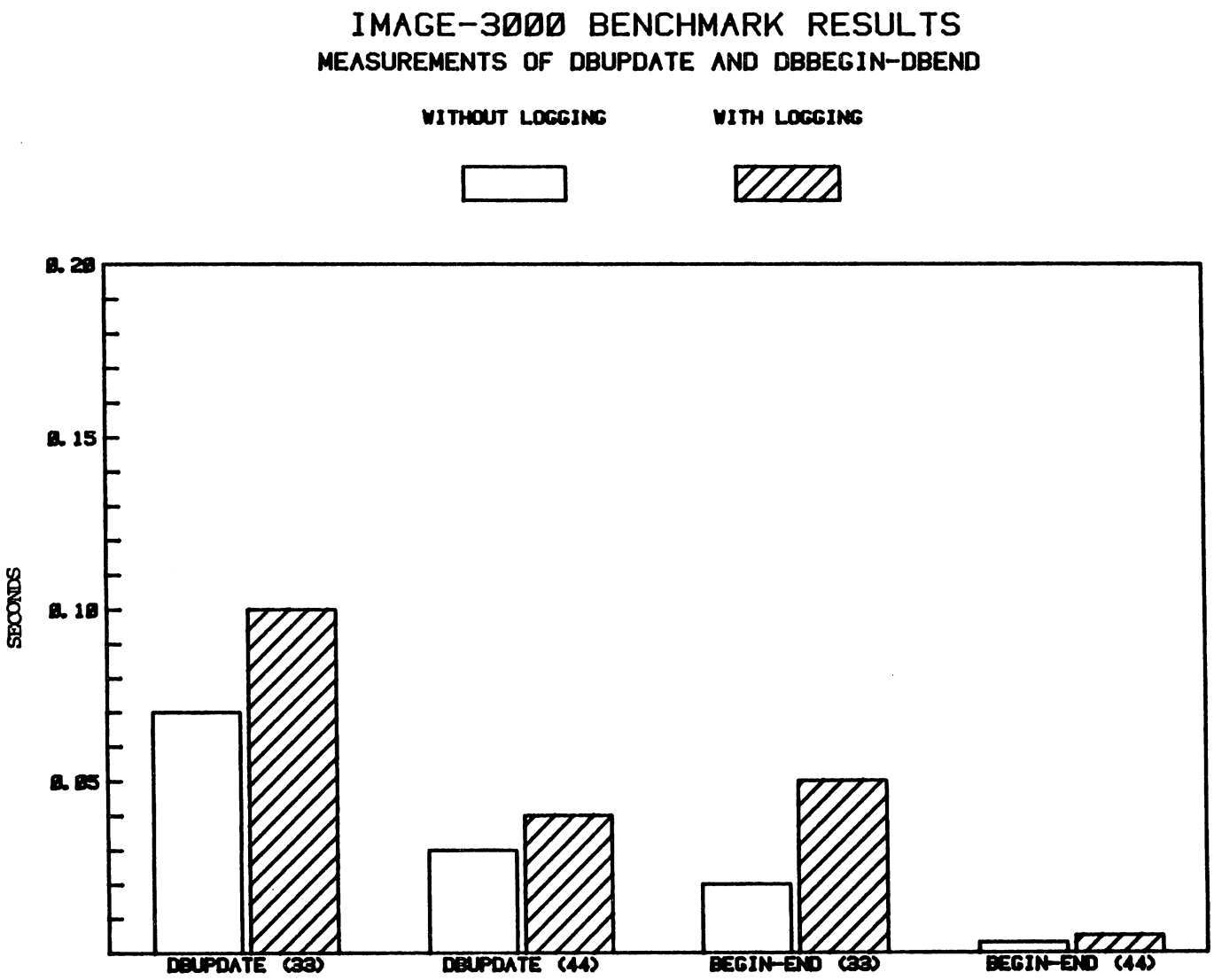


IMAGE-3000 BENCHMARK RESULTS
MEASUREMENTS OF DBUPDATE AND DBBEGIN-DBEND
WITHOUT LOGGING
WITH LOGGING
0.20
0.15
0.10
0.05
SECONDS
DBUPDATE (33)
DBUPDATE (44)
BEGIN-END (33)
BEGIN-END (44)

Figure 7

```
CALL DBBEGIN(BASE,...   )
     CALL DBLOCK(BASE,...  )

     CALL DBDELETE(BASE,... ) <structural damage,if crash occurrs>
                              < for a detail set with 5 paths >
                              < the 'critical' time could be  >
                              < a half second or more!        >

               o                      At this point,
               o                   if there is a crash we lose
               o                   this data entry!
               o

      change made to search item

               o
               o
               o

     CALL DBPUT(BASE,... ) <structural damage, if crash occurrs>
                              < for a detail set with 5 paths >
                              < the 'critical' time could be  >
                              < a half second or more!        >

               o                   The item has now be re-added

     CALL DBUNLOCK(BASE,... )
CALL DBEND(BASE,...   )
```

Figure 8. Crash Modes

Benefits of Logging VS Cost of Logging

AUDIT TRAIL
Who, What, When and How
Ability to list Sets and fields which are modified.

RECOVERABLE DATA
Ability to recover most if not all transactions, upto the time of the crash.

PERFORMANCE INFO
Information available which can lead to better application designs in the future.

LOGGING OF ALL CHANGES MADE TO DATA BASE REGARDLESS OF PROGRAM
You can use any vendor software and still maintain an "audit trail".

HP SUPPORT OF LOGGING

REDUCED THROUGHPUT?
Dependent on your application, and system load.

COST OF ADDITIONAL MEMORY?
May need more memory to maintain current system throughput.

TAPE DRIVE OR DISC DEDICATED TO LOGGING?
Valuable disc space or tape drive can be tied up with logging.

STARTUP AFTER CRASH MORE COMPLICATED

Training and "test recoverys" may be required to familiarize the programmers and operators with the new system restart procedures.

Figure 9

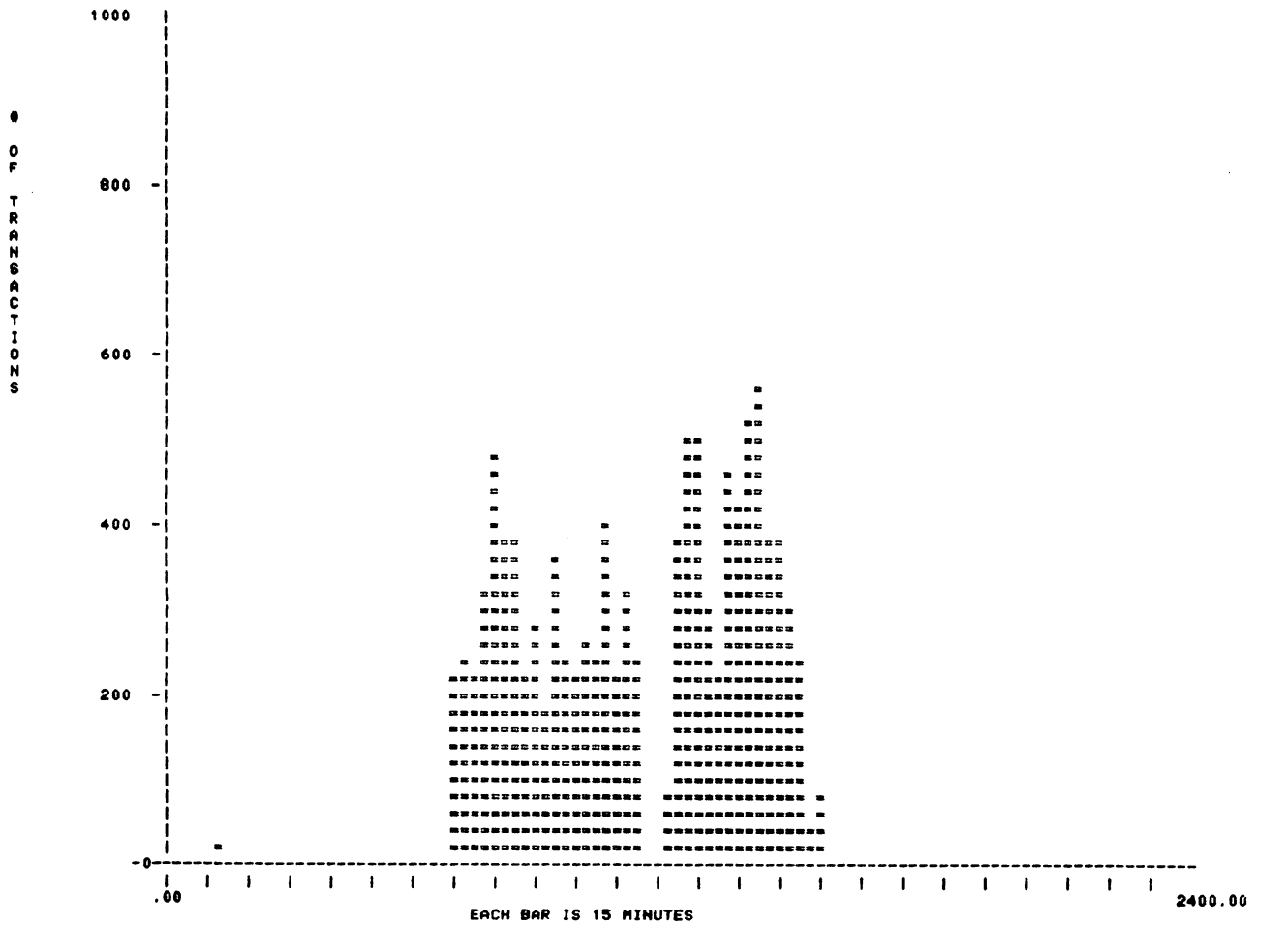
```
#     OFFICE  BAC      TEIMS          ACCTPROGBAC      TEIMS          TUE, DEC 29, 1981,  8:58 PM                          #
# 1                     ZmBBCE                                                                                              #
#                                                                                                                           #
#          LOGID#:   1  TRANSACTION# -     4   DELETEING ITEM IN      EQUIP-DETL         DBFILE RECORD#: 23277             #
#     PROP#          [U12  ] = 1                                                                                            #
#     MODELCOD       [U14  ] = 0003FIDDLE                                                                                   #
#     NOMNCODE       [I1   ] = 3                                                                                            #
#     EQUIPLOC       [I1   ] = 10                                                                                           #
#     PROGTAG        [I1   ] = 0                                                                                            #
#     CURRUSER       [U10  ] = 0000000100                                                                                   #
#     NEXTUSE1       [U10  ] = NONE                                                                                         #
#     PO-SERIES      [U10  ] = NONE                                                                                         #
#     HOLDFOR        [U10  ] = NONE                                                                                         #
#     BORROWER       [U10  ] = NONE                                                                                         #
#     FROMUSER       [U10  ] = NONE                                                                                         #
#     MFG            [I1   ] = 0                                                                                            #
#     INVTORYTAG     [I1   ] = 0                                                                                            #
#     CALDATE        [I2   ] = 811225                                                                                       #
#     SERIAL#        [U10  ] = SERIAL                                                                                       #
#     ACQUCOST       [I2   ] = 11                                                                                           #
#     CAPEXPEN       [U2   ] = E                                                                                            #
#     NEWUSED        [U2   ] = N                                                                                            #
#     ISSUEDAT       [I2   ] = 811220                                                                                       #
#     ADCODE         [U2   ] = AC                                                                                           #
#     ACQUDATE       [I2   ] = 811115                                                                                       #
#     NEXTSTD1       [I2   ] = 0                                                                                            #
#     NEXTSTD2       [I2   ] = 0                                                                                            #
#     HOLDATE        [I2   ] = 0                                                                                            #
#     NEXTEND1       [I2   ] = 0                                                                                            #
#     NEXTEND2       [I2   ] = 0                                                                                            #
#     STARTCYCLE     [I2   ] = 0                                                                                            #
#     RETNDATE       [I2   ] = 0                                                                                            #
#     BORROWDT       [I2   ] = 0                                                                                            #
#     MESSTAG1       [I1   ] = 0                                                                                            #
#     MESSTAG2       [I1   ] = 0                                                                                            #
#     MESSTAG3       [I1   ] = 0                                                                                            #
#     ACCESTAG       [I1   ] = 0                                                                                            #
#     OPTINTAG       [X2   ] = AA                                                                                           #
#     SPECCODE       [I1   ] = 0                                                                                            #
#     BAMPUTIL       [U2   ] = YE                                                                                           #
#     LASTUSER       [U10  ] = 5176822150                                                                                   #
#          LOGID#:   1  TRANSACTION# -     4   ADDING ITEM TO       EQUIP-DETL         DBFILE RECORD#: 23277             #
#     PROP#          [U12  ] = 1                                                                                            #
#     MODELCOD       [U14  ] = 0003FIDDLE                                                                                   #
#     NOMNCODE       [I1   ] = 3                                                                                            #
#     EQUIPLOC       [I1   ] = 10                                                                                           #
#     PROGTAG        [I1   ] = 0                                                                                            #
#     CURRUSER       [U10  ] = 5172870970                                                                                   #
#     NEXTUSE1       [U10  ] = NONE                                                                                         #
#     PO-SERIES      [U10  ] = NONE                                                                                         #
#     HOLDFOR        [U10  ] = NONE                                                                                         #
#     BORROWER       [U10  ] = NONE                                                                                         #
#     FROMUSER       [U10  ] = NONE                                                                                         #
#     MFG            [I1   ] = 0                                                                                            #
#     INVTORYTAG     [I1   ] = 30003                                                                                        #
#     CALDATE        [I2   ] = 811225                                                                                       #
#     SERIAL#        [U10  ] = SERIAL                                                                                       #
#     ACQUCOST       [I2   ] = 11                                                                                           #
#     CAPEXPEN       [U2   ] = E                                                                                            #
#     NEWUSED        [U2   ] = N                                                                                            #
#     ISSUEDAT       [I2   ] = 811220                                                                                       #
#     ADCODE         [U2   ] = AC                                                                                           #
#     ACQUDATE       [I2   ] = 811115                                                                                       #
#     NEXTSTD1       [I2   ] = 0                                                                                            #
#     NEXTSTD2       [I2   ] = 0                                                                                            #
#     HOLDATE        [I2   ] = 0                                                                                            #
#     NEXTEND1       [I2   ] = 0                                                                                            #
#     NEXTEND2       [I2   ] = 0                                                                                            #
#     STARTCYCLE     [I2   ] = 0                                                                                            #
#     RETNDATE       [I2   ] = 0                                                                                            #
#     BORROWDT       [I2   ] = 0                                                                                            #
#     MESSTAG1       [I1   ] = 0                                                                                            #
#     MESSTAG2       [I1   ] = 0                                                                                            #
#     MESSTAG3       [I1   ] = 0                                                                                            #
#     ACCESTAG       [I1   ] = 0                                                                                            #
#     OPTINTAG       [X2   ] = AA                                                                                           #
#     SPECCODE       [I1   ] = 0                                                                                            #
#     BAMPUTIL       [U2   ] = YE                                                                                           #
#     LASTUSER       [U10  ] = 5176822150                                                                                   #
#                    8:58 PM                                                                                                #
#############################################################################################################################

#############################################################################################################################
#     OFFICE  BAC      TEIMS          ACCTPROGBAC      TEIMS          TUE, DEC 29, 1981,  8:59 PM                          #
#                                                                                                                           #
#                                                                                                                           #
#          LOGID#:   1  TRANSACTION# -     5   UPDATING ITEM IN      EQUIP-DETL         DBFILE RECORD#: 23277             #
#     NEW VALUES:                                                                                                           #
#     CALDATE        [I2   ] = 0                                                                                            #
#     ADCODE         [U2   ] = NA                                                                                           #
#                                                                                                                           #
#     OLD VALUES:                                                                                                           #
#     CALDATE        [I2   ] = 811225                                                                                       #
#     ADCODE         [U2   ] = AC                                                                                           #
#                    8:59 PM                                                                                                #
#############################################################################################################################

          LOGID#:   1  TRANSACTION# -     1   UPDATING ITEM IN      EQUIP-DETL         DBFILE RECORD#: 23277
     NEW VALUES:
     MFG            [I1   ] = 3

     OLD VALUES:
     MFG            [I1   ] = 0
```

Figure 10

USER	GROUP	ACCT	DBASE	PROCESS	GROUP	ACCT	LOGON TIME	LOGOFF TIME	LG#	DEV	O	CAPABILITY	UP	PUT	DEL	#BLKS
OFFICE	BAC	TEIMS	TEIM1	QUERY	PUB	SYS	WED,NOV 25,1981, 1:55P	NOV 25, 2:02P	163	25	1	4020300611	1	0	0	0
BML	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 1:50P	NOV 25, 2:04P	162	37	1	0060300601	9	3	3	3
	WED, NOV 25, 1981, 2:13 PM		*** PROCESS ABORTED ***		PLTII	BAC	TEIMS									
PLTII	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:05P	NOV 25, 2:13P	164	36	1	0020300611	3	1	1	1?
OFFICE	BAC	TEIMS	TEIM1	QUERY	PUB	SYS	WED,NOV 25,1981, 2:07P	NOV 25, 2:14P	165	25	1	4020300611	1	0	0	0
DC	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 8:01A	NOV 25, 2:17P	132	34	1	0020300611	98	50	38	38
BAC	BAC	TEIMS	TEIM1	ACCTST	BAC	TEIMS	WED,NOV 25,1981, 2:19P	NOV 25, 2:20P	169	26	1	5360300613	0	0	0	0
DC	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 6:55A	NOV 25, 2:36P	127	39	1	0020300611	114	54	42	42
PRIMARY	BAC	TEIMS	TEIM1	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:09P	NOV 25, 2:38P	166	37	1	0020300611	18	6	6	6
PRIMARY	BAC	TEIMS	TOOLS	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:09P	NOV 25, 2:38P	167	37	1	0020300611	0	0	0	0*
QA	BAC	TEIMS	TEIM1	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 6:32A	NOV 25, 2:52P	122	32	1	0020300611	251	99	99	99
QA	BAC	TEIMS	TOOLS	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 6:32A	NOV 25, 2:52P	123	32	1	0020300611	0	0	0	0*
QA	BAC	TEIMS	TEIM1	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:53P	NOV 25, 2:56P	172	32	1	0020300611	0	0	0	0
QA	BAC	TEIMS	TOOLS	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:53P	NOV 25, 2:56P	173	32	1	0020300611	0	0	0	0*
QA	BAC	TEIMS	TEIM1	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:58P	NOV 25, 2:59P	174	32	1	0020300611	0	0	0	0
QA	BAC	TEIMS	TOOLS	QAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:58P	NOV 25, 2:59P	175	32	1	0020300611	0	0	0	0*
BAC	BAC	TEIMS	TEIM1	QUERY	PUB	SYS	WED,NOV 25,1981, 2:43P	NOV 25, 3:04P	171	26	1	5360300613	0	2	1	0
PLTII	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:16P	NOV 25, 3:05P	168	36	1	0020300611	49	23	18	18
KENT	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 6:39A	NOV 25, 3:08P	124	33	1	0020300611	220	97	78	83
DC	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 2:37P	NOV 25, 3:20P	170	34	1	0020300611	16	8	6	6
KENT	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 6:26A	NOV 25, 3:24P	120	31	1	0020300611	185	99	71	71
KENT	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 3:09P	NOV 25, 3:26P	177	32	1	0020300611	12	8	5	5
BAC	BAC	TEIMS	TEIM1	ACCTPROG	BAC	TEIMS	WED,NOV 25,1981, 3:04P	NOV 25, 3:39P	176	26	1	5360300613	4	1	1	4
	WED, NOV 25, 1981, 3:44 PM		*** PROCESS ABORTED ***		TEKSTAFF	BAC	TEIMS									
TEKSTAFF	BAC	TEIMS	TEIM1	TECHPROG	BAC	TEIMS	WED,NOV 25,1981,12:58P	NOV 25, 3:44P	159	21	1	0020300611	109	20	1	98?
OFFICE	BAC	TEIMS	TEIM1	ACCTPROG	BAC	TEIMS	WED,NOV 25,1981,12:13P	NOV 25, 3:47P	152	25	1	4020300611	136	76	73	212
BAC	BAC	TEIMS	TEIM1	ACCTST	BAC	TEIMS	WED,NOV 25,1981, 3:55P	NOV 25, 4:00P	178	26	1	5360300613	0	0	0	0
BAC	BAC	TEIMS	TEIM1	HAPROG	BAC	TEIMS	WED,NOV 25,1981, 4:27P	NOV 25, 4:45P	179	31	1	5360300613	0	0	0	0

* - INDICATES PROCESS DID NOT QUALIFY IN SELECTIVE SEARCH ? - BROKEN TRANSACTION

Figure 11



MAXIMUM VALUE: 560.0 MIN IS : .0 SCALE FACTOR: 20.0 AVG : 111.86 Y AXIS MAX: 1000 TOTAL # IN ALL CELLS: 11186

Figure 12

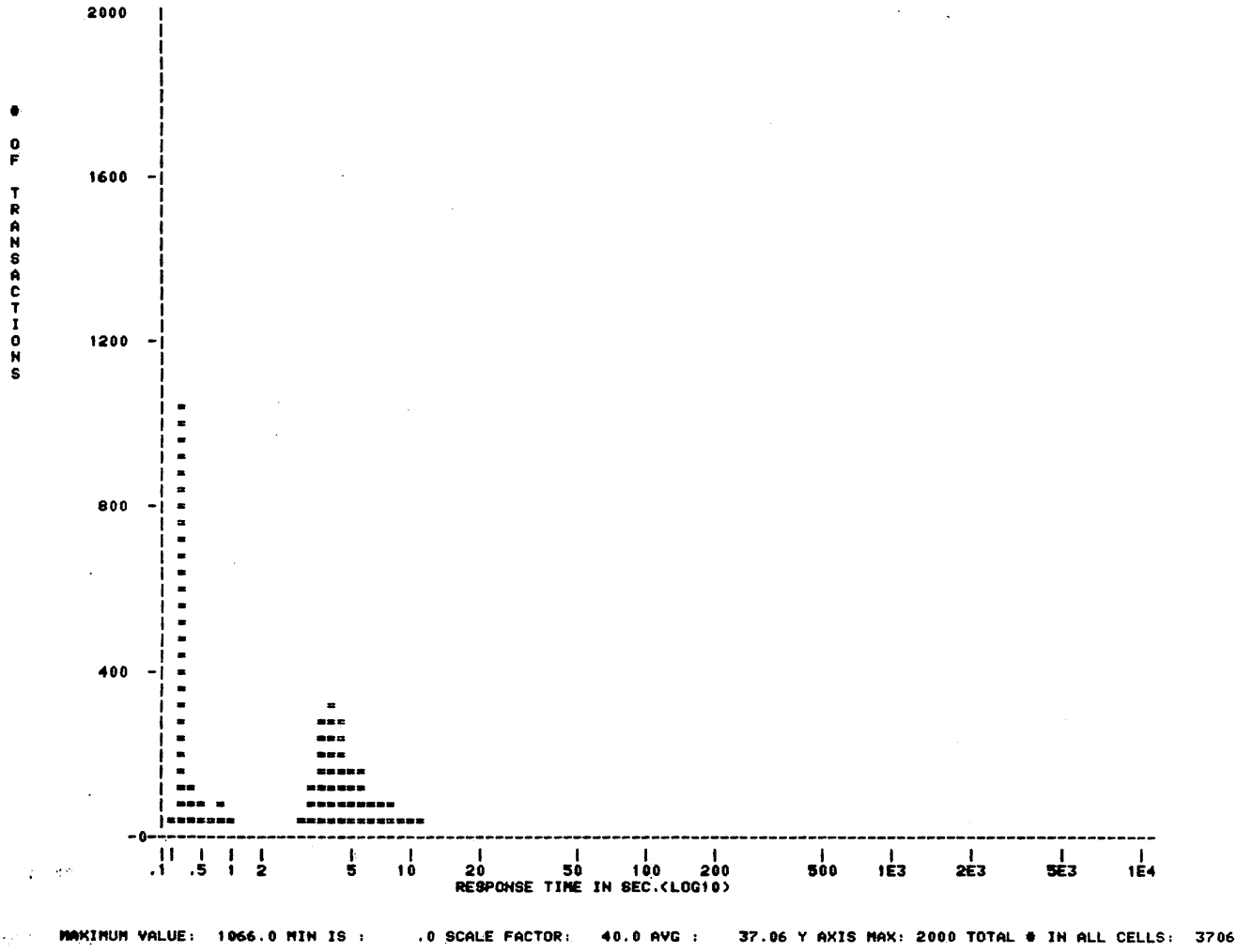

2000
1600
1200
800
400
0
OF TRANSACTIONS
.1
.5
1
2
5
10
20
50
100
200
500
1E3
2E3
5E3
1E4
RESPONSE TIME IN SEC.(LOG10)
MAXIMUM VALUE: 1066.0 MIN IS : .0 SCALE FACTOR: 40.0 AVG : 37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS: 3706

Figure 13

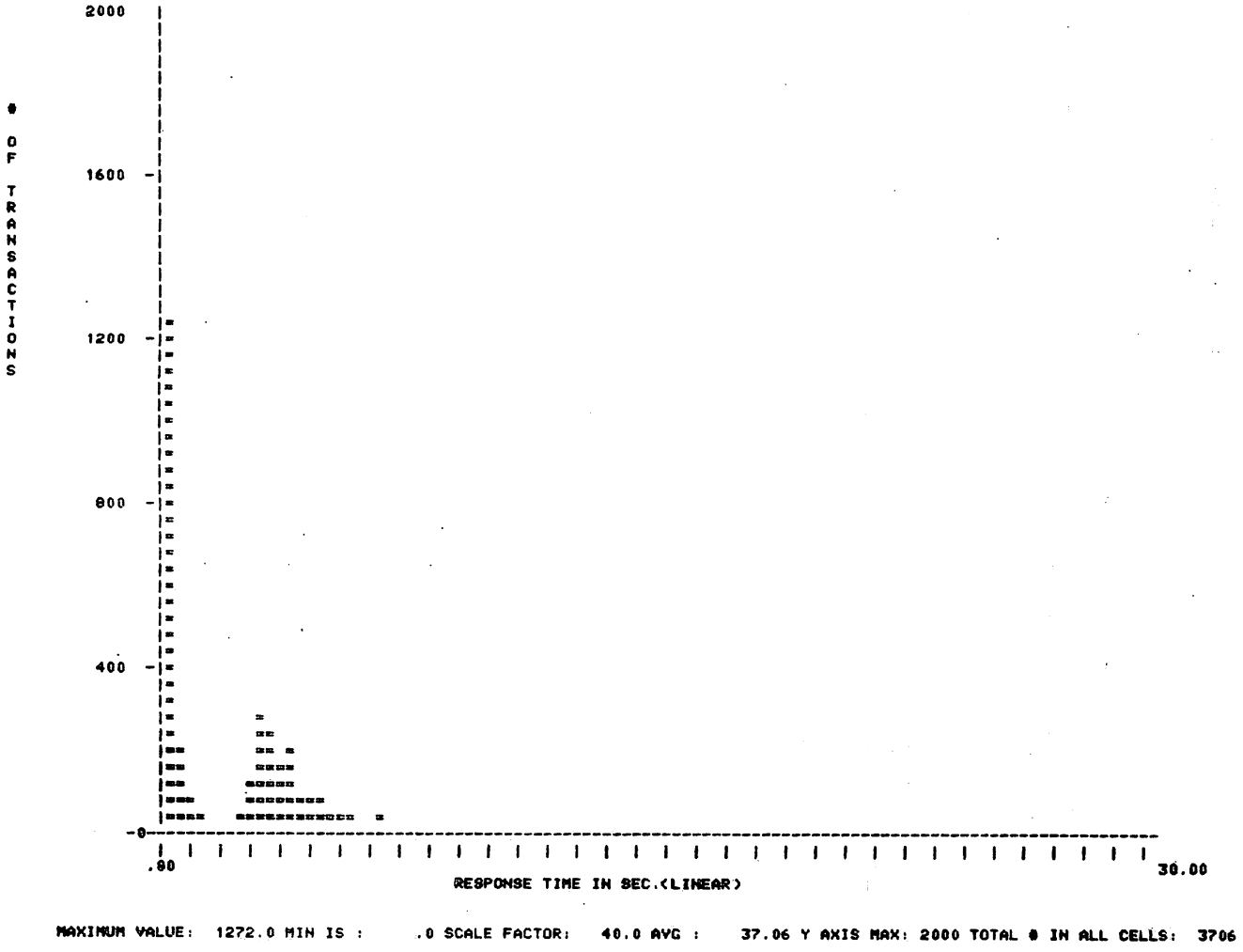

OF TRANSACTIONS
2000
1600
1200
800
400
0
.00
30.00
RESPONSE TIME IN SEC.(LINEAR)
MAXIMUM VALUE: 1272.0 MIN IS : .0 SCALE FACTOR: 40.0 AVG : 37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS: 3706

Figure 14

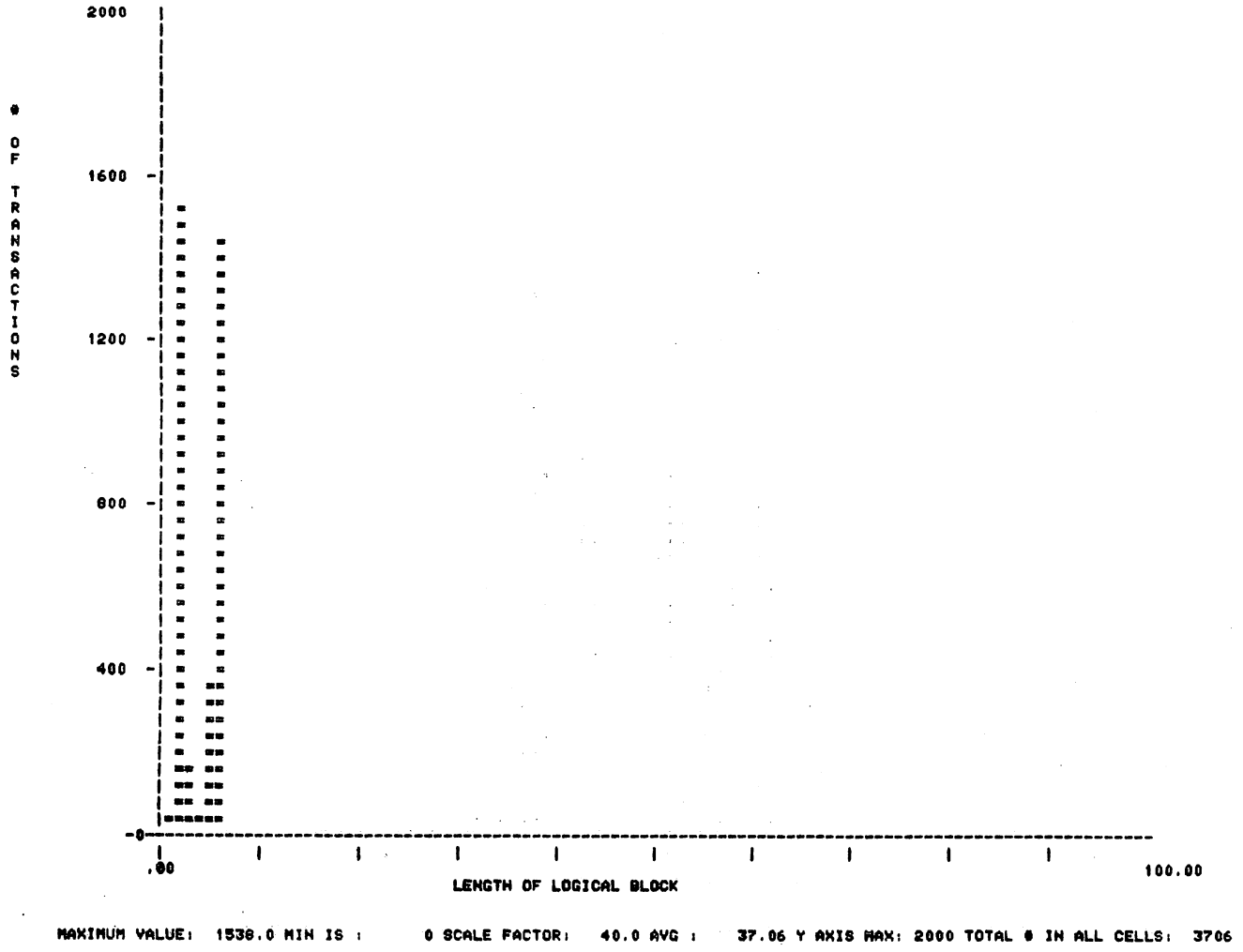

2000
1600
1200
800
400
0
OF TRANSACTIONS
.00
100.00
LENGTH OF LOGICAL BLOCK
MAXIMUM VALUE: 1538.0 MIN IS : 0 SCALE FACTOR: 40.0 AVG : 37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS: 3706

Figure 15

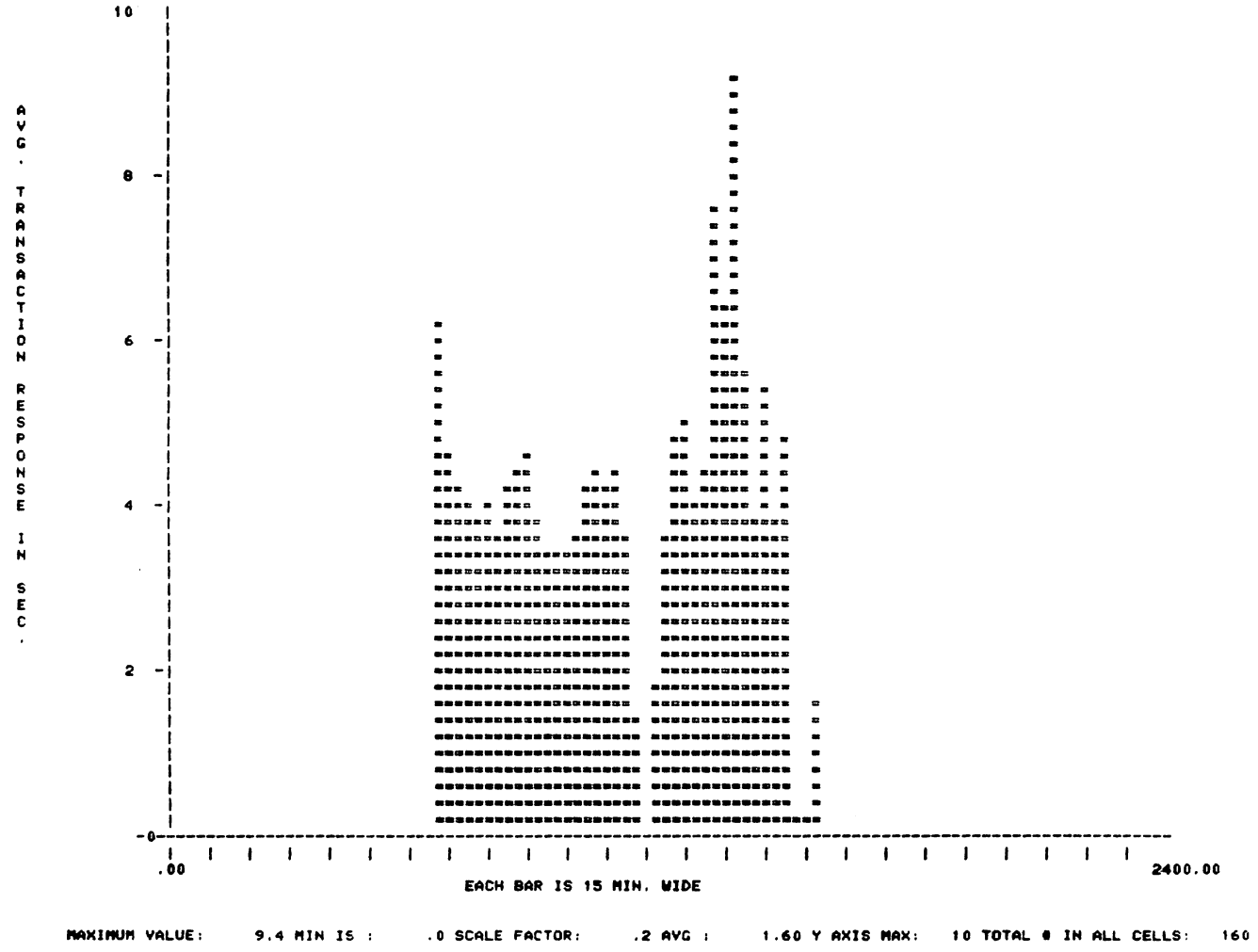

AVG. TRANSACTION RESPONSE IN SEC.
10
8
6
4
2
0
.00
2400.00
EACH BAR IS 15 MIN. WIDE
MAXIMUM VALUE: 9.4 MIN IS : .0 SCALE FACTOR: .2 AVG : 1.60 Y AXIS MAX: 10 TOTAL # IN ALL CELLS: 160

Figure 16

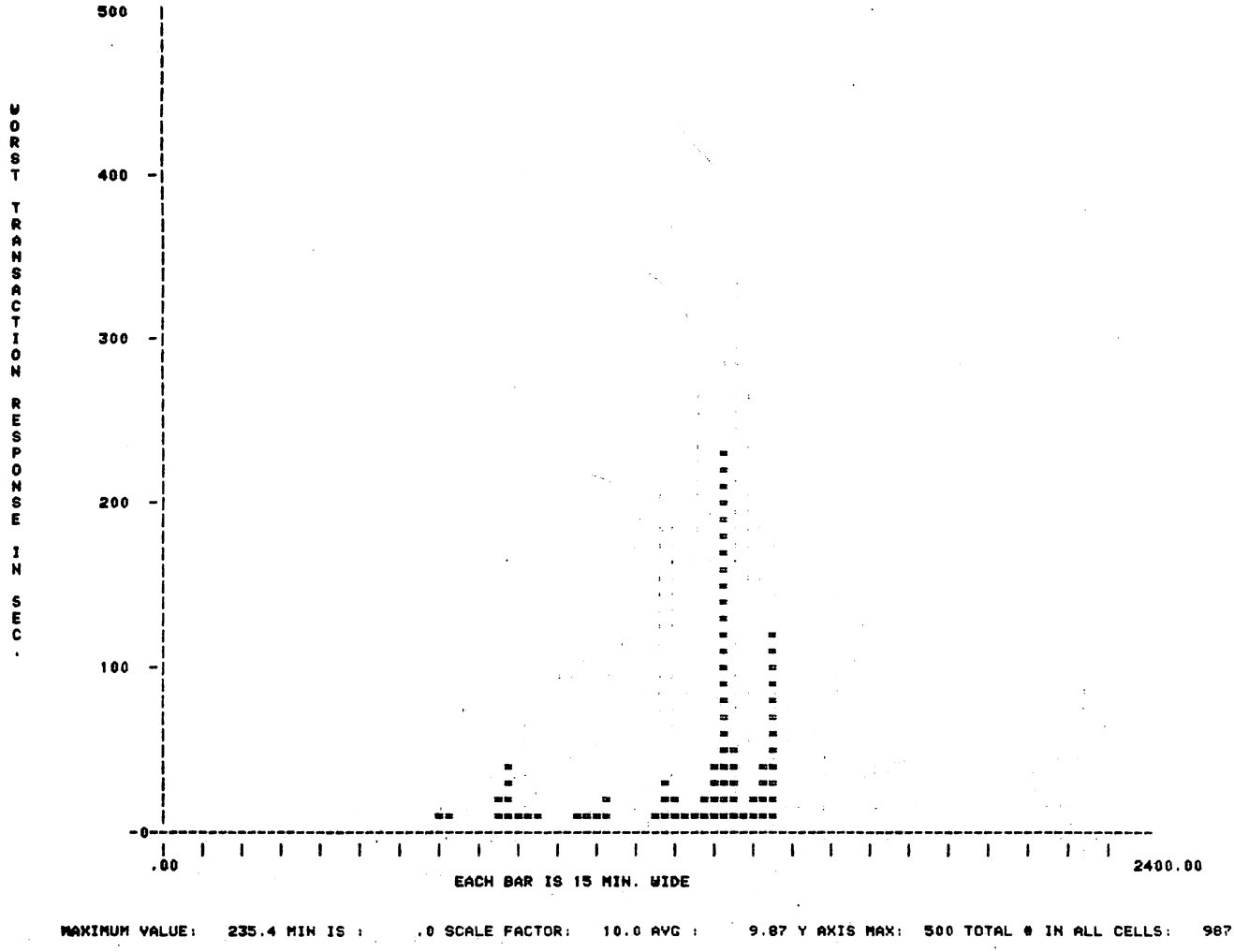

WORST TRANSACTION RESPONSE IN SEC.
500
400
300
200
100
-0
.00
2400.00
EACH BAR IS 15 MIN. WIDE
MAXIMUM VALUE: 235.4 MIN IS : .0 SCALE FACTOR: 10.0 AVG : 9.87 Y AXIS MAX: 500 TOTAL # IN ALL CELLS: 987

Figure 17

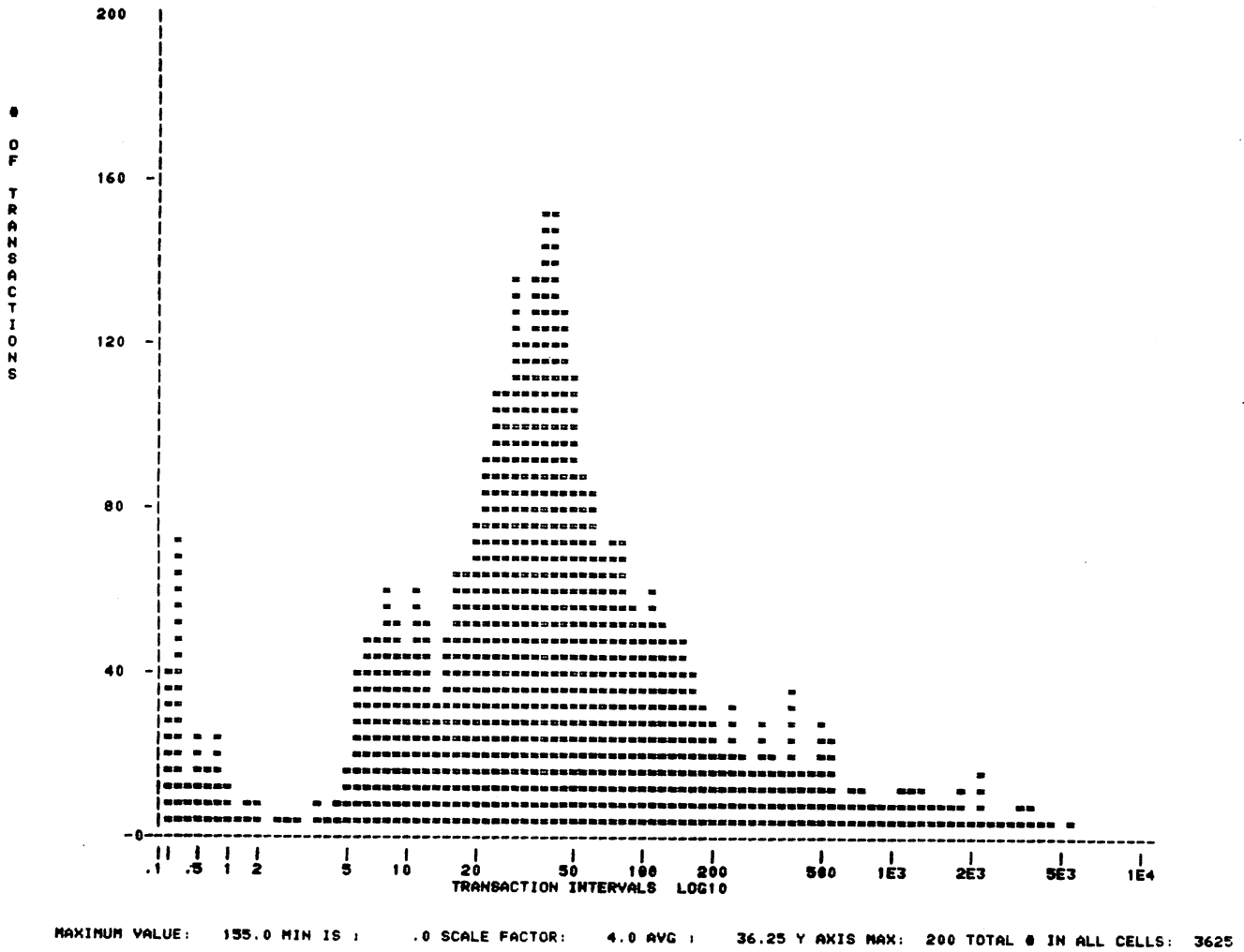

200
160
120
80
40
0
OF TRANSACTIONS
.1
.5
1
2
5
10
20
50
100
200
500
1E3
2E3
5E3
1E4
TRANSACTION INTERVALS LOG10
MAXIMUM VALUE: 155.0 MIN IS : .0 SCALE FACTOR: 4.0 AVG : 36.25 Y AXIS MAX: 200 TOTAL # IN ALL CELLS: 3625

Figure 18

```
**************************************************************
*      NUMBER OF RECORDS PROCESS          22112              *
*      PUTS, DELETES, UPDATES             11186              *
*      DBBEGINS & DBENDS                   7415              *
*      AVERAGE TRANSACTION TIME            1.60              *
*                STD DEVIATION             4.57              *
*                                                            *
*      AVERAGE TRANSACTION INTERVAL      182.04              *
*            STD DEVIATION               645.68              *
*                                                            *
*      AVERAGE BLOCK LENGTH                1.49              *
*            STD DEVIATION                 2.08              *
*      # OF LOGICAL BLOCKS                 3625              *
**************************************************************
```

Figure 19

DATA-SET(BASE)	#UPDATES	#DELETES	#PUTS	CAPACITY	ENTRIES	PERCENT FULL
CROSSREFJ05	0	8	7	50036	41355	82.65
OPTION-DETL	0	0	1	987	369	37.39
SERV-DETL	210	120	129	30984	21705	70.05
UTIL-DETL	759	45	532	30990	7443	24.02
EQUIP-DETL	704	2015	2021	34986	27363	78.21
NOMCL-DETL	399	1	28	5004	3970	79.34
USER-DETL	3732	0	0	1704	1140	66.90
NOMH-DETL	12	1	13	1512	697	46.10
SPEC-DETL	229	5	48	34986	16389	46.84
USEWITH-XREF	0	10	106	1014	101	9.96
WHAREHOUSELOC	0	24	27	5031	2518	50.05

Figure 20

RAPID/3000

Nancy Calwell
Hewlett-Packard

OVERVIEW

RAPID/3000 is a new product that has been introdued by Hewlett-Packard. The RAPID/3000 product package contains four components. Together, these components make up a powerful productivity tool when used in conjunction with existing HP3000 application development software. This product increases the productivity of both programmers and end users by effectively separating their needs for data. Thus, the database administrator and the end user are provided with a set of tools to solve each of their respective data processing needs.

The components of RAPID/3000 are broken down as follows:

- HP TRANSACT/3000
- HP REPORT/3000
- HP INFORM/3000
- HP DICTIONARY/3000
  - Dictionary/Directory
  - utilities

The remainder of this discussion will deal with a more comprehensive view of the four RAPID/3000 components.

TRANSACT/3000

TRANSACT/3000 is a high level programming language that allows access to all MPE data files. It has the ability to call subroutines written in COBOL, PASCAL, SPL, and even TRANSACT. The flexibility exists to call MPE intrinsics. TRANSACT supports all of the data types that exist on the HP3000. It also has a "built-in" interface to VPLUS. A single TRANSACT command has the ability to replace many VPLUS intrinsics that currently the programmer must code.

Because of the ability to quickly code solutions with TRANSACT, the area of "prototyping" is a natural for this language. Feedback can be given to the application designer before a lot of time is spent on a possible miscommunication.

There is a feature in TRANSACT/3000 that allows dynamic debug capabilities while designing or maintaining a program. System and program errors are returned to the program for local handling with no need to return to the system environment.

REPORT/3000

REPORT/3000 is a nonprocedural report writer. This means that the statements within the report may be coded in any order. REPORT has the capability to retrieve information from IMAGE, KSAM, and MPE files. Now simple reports can be produced by using the defaults of REPORT/3000. By using the more advanced commands, customized reports can be written.

INFORM/3000

INFORM/3000 is a menu driven report generator. It is geared towards the entire company by providing an ad hoc inquiry ability to even the most unsophisticated user. Additionally, INFORM/3000 allows a relational type access to any IMAGE, KSAM, or MPE file that is defined in the Dictionary.

DICTIONARY/3000

The Dictionary is an IMAGE data structure that contains information about your production data. It provides a single place to go for data physical attributes, responsibility, descriptions, and various other forms of documentation.

Information about your data can either be interactively entered or automatically transferred from existing IMAGE root files into the Dictionary. Reports on the Dictionary contents into the Dictionary contents can easily be generated by using Dictionary commands.

DICTIONARY/3000 supports user views of data. In addition to documenting the physical nature of the information system, the DICTIONARY also documents and makes available through Inform, relational views of the data.

Included in the Dictionary package is a set of utilities that assist in the maintenance of IMAGE databases.

A Data Dictionary and Directory Facility

Have any of the following situations happened to you?

You have been using a centrally maintained database. All of a sudden, a program that has been running successfully for six months fails. The reason? Over the weekend, the database administrator changed the definition of a data element. You were not notified.

You are responsible for a centrally maintained database. You are requested by the Information Systems Manager to change the definition of a data element. You proceed to run yourself ragged locating all of the users that will be affected by this change. After a week of playing Sherlock Holmes, the change is done.

Monday morning you find ten angry users who were not notified.

You spend months developing a series of programs only to find out that a large part of the data produced by the new system is already available.

You find that the space on the system is starting to be worth more than a troy ounce of gold. Upon investigation of the problem, you find there are several files that look conspicuously alike. The problem? Data redundancy.

The problems mentioned above are only a few of those that annoy and waste the valuable time of data processing people every day. DICTIONARY/3000 is the answer not only to the problems above, but to many situations that become problems in the wonderful world of data processing.

What is DICTIONARY/3000?

A dictionary must supply the answers to the questions

Who . . . ?
What . . . ?
Where . . . ?
When . . . ?

WHO is responsible for the data elements and files.

WHAT is their physical properties.

WHERE is their location.

WHEN is their usage.

The Dictionary should not only be used as a reference document, it is a tool to simplify documentation. It contains the description and directory information of the data. The data itself is still managed by IMAGE, KSAM, and MPE. Additionally, DICTIONARY/3000 may be used to describe systems, programs, subprograms, and procedures. It is also possible to document the entire corporate organization (i.e., an organizational chart). Data may be gathered into logical groups to quickly generate reports. This is done with the help of INFORM/3000. Also included with the purchase of DICTIONARY/3000 is a set of high powered utilities that assist in the maintenance of not only the Dictionary but production IMAGE databases as well.

When Should DICTIONARY/3000 Be Considered?

If you have a database, you should be considering DICTIONARY/3000. If you have a lot of data or a need to minimize the cost of application modification or development you should be considering DICTIONARY/3000. If you have a need to quickly link the data that resides in more than one IMAGE, KSAM, or MPE file. If you truly want to stop data redundancy. If you truly want to stop programmer time redundancy. If you have a HP3000.

A Closer Look

This section will deal more with the specifics on DICTIONARY/3000. The following outline indicates the topics that will be covered with respect to DICTIONARY/3000, IMAGE/3000 and other MPE files. The presentation will be slide/lecture and will span the full hour. Time will be given at the end of the presentation for a question and answer period.

I. DICTIONARY/3000
  A. A set of powerful tools
    1. Descriptive tool
      a. Composition of items
      b. What types of files use these items.
    2. Logistical tool
      a. Who uses which files
      b. Where are these files located
    3. Operational tool
      a. Databases can be created thru the Dictionary
      b. Dictionary entries can be created thru a database.
  B. Dictionary Structure
    1. Physical structure
    2. Logical structure
      a. physical entities (i.e., items, files, security)
      b. entity linkages
  C. Physical entities
    1. What are all of the physical entities available in the Dictionary
    2. How are the entities used?
      a. entities whose usage is program defined
      b. entities whose usage is user definable

II. DICTIONARY UTILITIES
  A. What are they
    1. A list of the utilities
    2. A closer look at what they do
  B. IMAGE utilities vs. DICTIONARY utilities
    1. IMAGE utilities still needed
    2. IMAGE utilities possibly replaced

Information Management: An Investment for the Future

David C. Dummer
IMACS Systems Corporation
Los Angeles, California

We are all witnesses to the current information explosion that affects every aspect of our lives. Some of us may well wonder if this explosion can be contained and controlled.

Computer technology has nurtured the evolution of devices that perform data storage and manipulation functions at reasonable costs. Government, industry and commerce have rapidly made use of such devices in an effort to improve information systems for decision-making processes. The better the quality and timeliness of information, the more powerful and competitive the user can become.

Unfortunately, the technologies that support the effective utilization of information systems have trailed the dramatic advances in computer hardware and software. There is still not a general awareness that data is an organizational resource that requires management control, administration and involvement. Figure 1 draws an analogy to other resource management areas in an organization; namely that good data management will directly benefit information systems needed for decision-making. Data management is, however, a far less developed area than either financial or personnel management. Very few organizations that have attempted to establish a corporate database resource have been completely successful. The history of integrated management information systems contains many failures and, in some cases, the downfall of the organization.

Like many technological advances, those related to information handling are full of promise, yet also hide many dangers and pitfalls. Failures can generally be attributed to two major causes: the incomplete and incorrect application of the technologies and resulting information handling facilities; and the lack of necessary changes to the organizational structure to complement the integrated information structure.

It is relatively easy for senior executives to decide upon a data resource management strategy, but it is a far different matter to understand all of the different components necessary for the success of the strategy. These problems are compounded by the fact that there are still very few professionals in this area with adequate levels of experience.

Figure 2 highlights the resistance that corporate management typically meets in the introduction of database technology. Resistance can occur in both data processing and user departments as the need for new responsibilities and procedures becomes evident. Few executives are equipped with all of the correct rebuttals to the criticisms that result from the resistance. So intense can the resistance become that often the database approach is introduced in a compromised manner and one that is far from being optimal for the organization.

Corporate opportunities that can be realized by the database approach are immeasureable and there is an ever increasing responsibility on the data processing profession to ensure that the approach is fully understood and supported. Figure 2 also enumerates the respective benefits that can accrue to the organization but, in order to achieve these benefits, the organization must be willing to invest the necessary developmental resources into the database approach.

Once the database approach has been adopted as a means of achieving data resource management, it is important to realize that a data sharing concept has been introduced within the organization. That is, the common data in the corporate database is to be shared by all those in the organization that have a right and need to access the data. The major technical facility that supports data sharing is a database management system (DBMS). Such a system is often presented as a solution to the problem of data sharing but, in reality, it is simply just one of the facilities needed to achieve data sharing in a resource management environment.

Crucial to the success of data sharing is data administration, sometimes referred to as database administration. Data administration encompasses other facilities, procedures and tools needed to manage the corporate database. Figure 3 shows the major aspects of data administration. The degree to which an organization addresses and implements facilities in these areas should depend on the complexity, integration and value of the data together with any reasonable limitations imposed by the budget available for data management.

First and foremost of the required data adminstration facilities is a data dictionary and directory (Dictionary). The Dictionary is essentially an information system about the data and data processing systems used in the organization. To the person or persons responsible for

data administration, it represents a tool to document and control the corporate database facility. If database driven information systems are an integral part of the operations of the organization then the database will normally have to be flexible and changeable in order to reflect and support the business dynamics. The Dictionary should be designed and organized to provide for this type of environment.

For some organizations the Dictionary will evolve into the hub, or nerve-center, of their data processing facilities. It will control, monitor and service the corporate database together with the associated information systems.

There are many other data administration facilities which complement the Dictionary. Some of them are still in the evolutionary stage but collectively they address such considerations as ensuring that the corporate database is always organized in the most efficient manner; is normally available for access; and is recoverable in the event of system failures and errors.

A further important aspect of information systems concerns auditability and performance measurement. These are typically topics that are considered only at system implementation but, a data administration function can ensure that system audit becomes a design parameter during system analysis and development.

Figure 4 shows the responsibitities of the person (Data Administrator) or persons performing the data administration function. Like other resource managers, the Data Administrator must be placed in an organizational structure such that he or she can be held responsible for all of the technical and administrative components needed to effect the data sharing. The Data Administrator is responsible to the corporate executives to inform them of matters and situations that demand their participation and decision-making; and then to enact and administer the resulting policies and data control measures. The Data Administrator is responsible to the data processing users in terms of responding to queries, service requests and the provision of facilities to make the data more accessible and useable by those having the right to share it. The Data Administrator interfaces with the systems group in order to obtain hardware and software required to provide and support the database and processing environment for which he or she is responsible. The fourth and final interface within the organization is to the corporate data management system which encompasses all of the facilities needed to provide the control and administration of the corporate database.

With a perception of these data administration requirements the specification of a suitable Dictionary can be detailed. Figure 5 lists the major components of a data dictionary and directory. The content and complexity of the Dictionary should again be a reasonable balance between the requirements and budget of the organization. The Dictionary contents cover the documentation of how data is perceived in the organization (documents, forms, usage by function or department, etc.); how data is structured in the database and file designs; and how data is used in the data processing systems. Supporting these directories are dictionary entries which document the attributes of the data processing components, specify any data validation and security rules, and detail any data processing component alias naming conventions.

Figure 6 depicts the role of the Dictionary in the organization. It is the tool by which the Data Administrator documents, controls and administers the corporate database and information systems. It is a source of information and a design capture facility for database and information system designers. It is a source of information and a change capture facility for the maintenance group as existing databases and information systems are modified and enhanced. It is a source of information to data processing users who can discover what data and processing systems are available without the need to contact the data administration staff. This is particularly true of an online Dictionary supporting ad-hoc user queries.

One data administration topic currently receiving a lot of attention is data security and privacy. Figure 7 contains a list of items needing protection considerations and a list of events that can constitute a threat to data and system security. The first list highlights the fact that security measures applied to databases and files are of little value if complementary security measures are not applied to the computer memory and storage devices used during data processing; the hard copy data listings and reports distributed within the organization; the data transmission lines used by remote terminals, work stations and computers; and the security measures themselves. In a similar manner to security provided by a lock and key, database and processing security is only a deterrent and each extra level of security will typically involve exponential increases in the cost and time of the security measure implementation and practice. These extra measures should only be applied where warranted by the sensitivity of the particular data or process.

The second list in figure 7 covers the major threats to the security and availability of the corporate database. These are important considerations in a data sharing environment since the data has been organized in a logically or physically central location and it is collectively more vunerable to security violation. One of the most crucial tasks for the Data Administrator is the achievement of a correct balance between data accessibility and security for the organization.

The challenge facing most organizations at this time is the effective creation of the data sharing environment. It requires an investment in terms of people, funds and organizational change; but the future benefits of a well managed data resource to aid and make possible decision-making are immeasurable.

The presentation of the paper will enlarge upon these

topics and suggest various methods of evaluating and implementing data administration facilities and procedures in the HP3000 computer system environment.

Figure 1. Resource Management

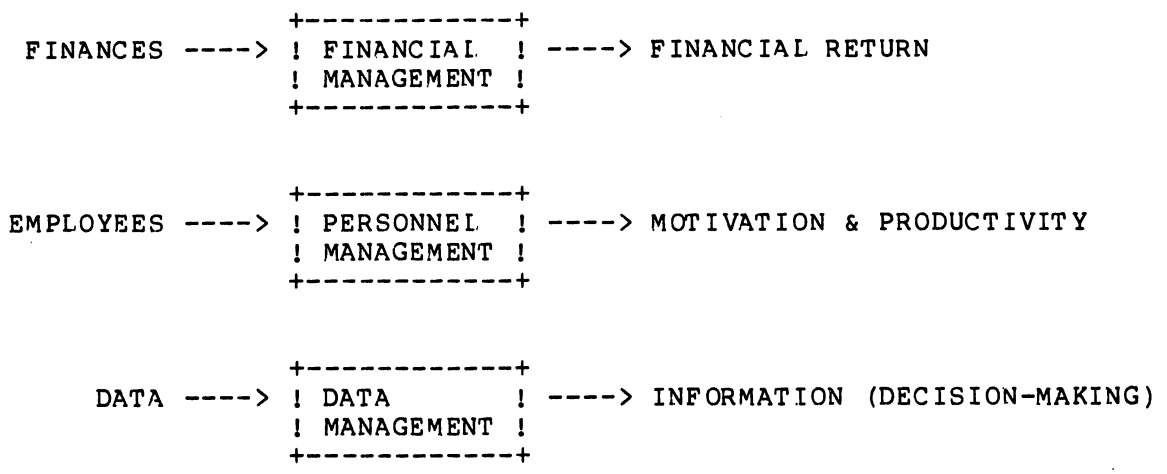


Figure 2. Introducing Database Technology to an Organization

RESISTANCE TO:	CORPORATE OPPORTUNITY:
- CHANGE IN METHODS AND PROCEDURES	- IMPROVED METHODS AND PROCEDURES
- LOSS OF DATA OWNERSHIP	- CORPORATE DATA MANAGEMENT
- LOSS OF DATA CONTROL	- CORPORATE CONTROL STANDARDS
- CHANGE IN POWER STRUCTURE	- PROFESSIONAL DATA MANAGEMENT
- CHANGE IN THE STATUS OF DATA PROCESSING USERS	- IMPROVED DATA UTILIZATION
- CHANGE IN STAFF REQUIREMENTS	- REDUCED COST AND ABILITY TO CONTROL

Figure 3. Data Sharing Requirements

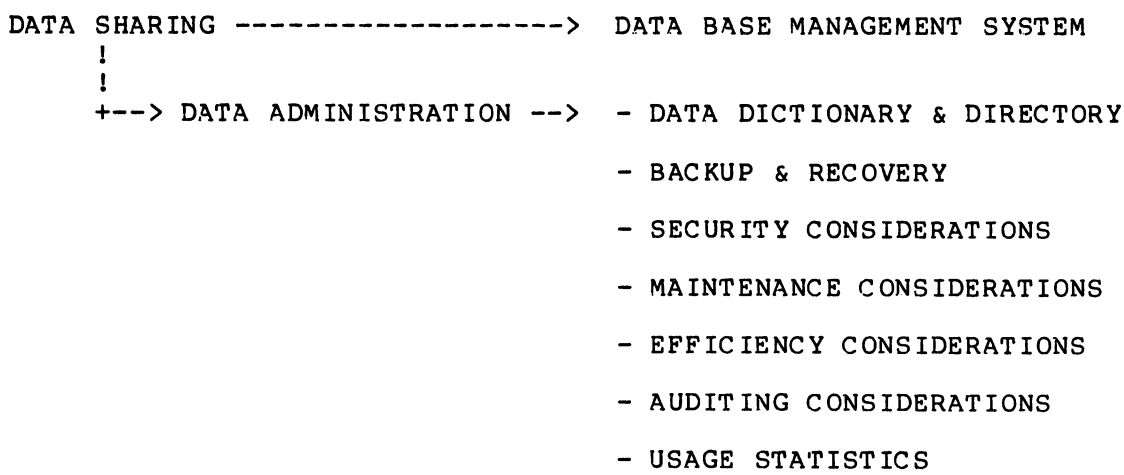


Figure 5. Role of the Data Administrator

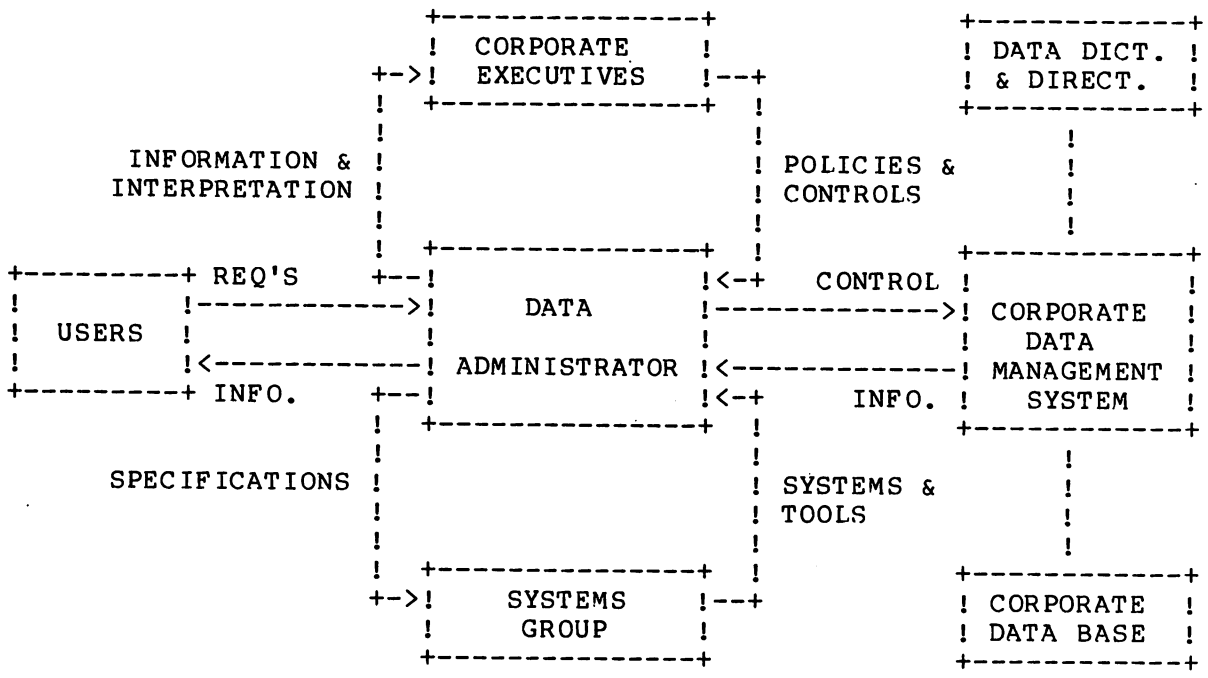
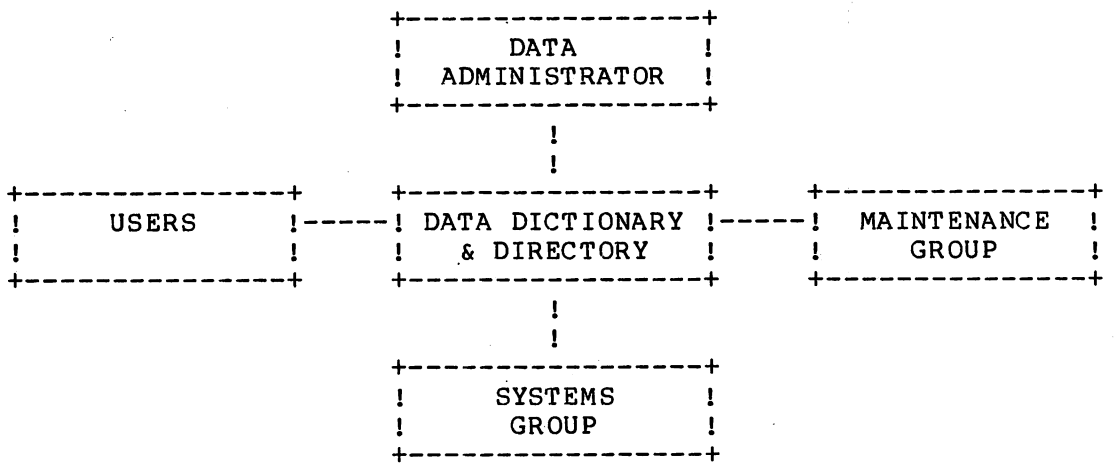


Figure 6. Data Dictionary & Directory Contents

- DICTIONARY & DIRECTORY OF THE NATURAL DATA ITEMS AND DATA ITEM GROUPINGS IN AN ORGANIZATION
- DICTIONARY & DIRECTORY OF THE DATA STRUCTURES DESIGNED FOR THE CORPORATE DATA BASE(S)
- DICTIONARY OF THE ATTRIBUTES OF DATA ITEMS, DATA FILES, DATA PROCESSES, ETC.
- DICTIONARY OF DATA ITEM VALIDATION RULES
- DIRECTORY OF DATA ITEM AND FILE SECURITY RULES
- DIRECTORY OF DATA ITEM SYNONYMS (ALIASES)
- DIRECTORY OF RELATIONSHIPS BETWEEN DATA ITEMS, DATA BASES, DATA FILES, DATA PROCESSES, DOCUMENTS, FUNCTIONS, PHYSICAL STORAGE DEVICES, ETC.

Figure 7. Role of the Dictionary & Directory



Successfully Developing On-Line RPG/3000 Applications

Duane Schulz
Hewlett-Packard Company
Wilsonville, Oregon

INTRODUCTION

Why do people laugh when I talk about interactive RPG/3000 applications? This paper will focus on the unique nature of RPG as an on-line language on the HP3000 computer system, identify the difficulties to be anticipated in learning how to best use the HP3000 with RPG as the primary language, and attempt to outline a clear path to success in this area, with attendant suppression of the abovementioned laughter. Hopefully, RPG users who review this paper will be able to identify and alleviate any current problems they are experiencing as interactive RPG users, and new users will be able to make the transition to the HP3000 smoothly.

To accomplish this, it will first be necessary to profile the probable assumptions of a new RPG/3000 user, outline the nature of the HP3000 computer, and identify any dissimilarities between the two. Once this is done, a series of steps designed to reconcile differences between RPG users' understanding of computers and the nature of the HP3000 will be presented. The reconciliation of these differences is the only true obstacle to success as an RPG/3000 user, though this basic fact is usually ignored while technical symptoms of the problem are addressed instead. the goal of the presentation is to help RPG users to finally come to grips with the struggle which is the inevitable result of conversion to the HP3000 while still relying on ideas and concepts which do not apply to the HP3000.

A USER PROFILE

In order to avoid stereotyping, I will describe my personal background at the time I first selected the HP3000 as a replacement to an IBM GSD computer. This background is very common, given the vast number of IBM System/3 and /34 systems currently installed. Also note that I'll assume that RPG/3000 users received their initial programming experience on IBM equipment; they set the de facto industry standard. The following statements summarize the computing concepts I employed at the time of my first conversion (please don't assume that these matched my philosophy about computing or my experience in actual implementation. . .):

- Transactions enter the computer (from the data processing department) in batches which must be 100% correct before the resulting data (usually a report) was returned to the user, again via data processing personnel.
- Each system was controlled by one user, and was 90% unrelated to other systems on the computer. Sharing common data was rare.
- Most technical solutions and learning was vendor-provided. User groups served other (social, system back-up) purposes.
- Interactive updates were best performed on dummy files, with the real updates performed during off-hours in a batch fashion.
- The system was best used for serial I/O.
- Multiple tasks were accomplished through fixed memory partitioning or special control programs (ie. CCP, etc.).
- Databases involve high overhead and long learning processes.
- Knowledge of operating system internals is not related to successful application development.
- Efficient programs were related to avoidance of high-overhead calculations and program logic.
- Data structures, screen handlers, and internals were best learned through RPG interfaces to such facilities. Knowledge of the specific facility on its own was not necessary (or, in some cases, possible).

HP3000: BASIC ASSUMPTIONS

Thought we all understand or have learned at least some of the following assumptions, it is important to note the nature of the HP3000 as compared with the mind set described above. Again, this is a list of statements or descriptions highlighting the important concepts behind the HP3000. These things must be learned before we can develop applications for the 3000 and expect them to be stable or efficient.

- The HP3000 is designed to be an interactive, user-driven computer, with transactions occurring randomly, usually with little involvement on the part of the data processing department.
- Most application systems will be (logically if not physically) related to one another.
- Learning and technical solutions are provided by

the vendor, a number of user groups, appropriate third-party assistance, and especially through self-teaching and exploration on the user's part.

- The system is designed for for interactive use. Serial I/O is one of the least efficient I/O techniques.
- The operating system includes a memory manager and dispatcher which allow dynamic memory sharing between users.
- IMAGE provides provides the most efficient data and I/O structure for the environments described above; VPLUS provides the most efficient method of communication with users (terminals). Both should be learned on their own, not through learning RPG interfaces.
- Knowledge of the file and operating system intrinsics is related to the success of applications.
- Efficient programs are related to efficient calculations and logic. RPG/3000 is externally driven, and there is no specific correlation between a given calculation and a specific set of machine instructions.

BRIDGING THE GAP

Understanding how to use RPG is the key to understanding how to use IBM GSD computers, because it was desgined to allow optimum use of those operating systems and instruction sets. This is clearly not the case with the HP3000. MPE was written as a language-independent (exception: SPL) operating system with independent constructs, as were IMAGE, VPLUS, KSAM and other subsystems. Obviously, our success in developing successful RPG/3000 applications lies, not as it did with IBM, in understanding MPE and its subsystems first, then learning how RPG interfaces with these things. The mistake made by the bulk of the RPG user community (at first) is to continue to rely on RPG as the window through which to peer into the computer; this is precisely what has earned us our reputation. The remainder of this paper will outline the steps involved in adopting MPE and its subsystems as language-independent constructs. Though this outline is not absolute, all of these steps must be taken in some fashion. There are no shortcuts that lead to anything other than unstable, costly to maintain systems.

1: Identify Your Resources

As early as possible, learn about any resources which are available to help you in completing the tasks outlined below. If you don't do this, being an HP RPG user will feel very lonely (let's face it, RPG is used by a minority of HP3000 customers). There are 4 sources of assistance:

HP: Read your support contract and understand what it buys you, and what is your own responsibility. If there are misunderstandings, clear them up before you proceed. Be sure your SE can help you with RPG learning and problems. He/she need not be an RPG expert to get you help. Find out who the closest RPG SE is, and arrange a path to that specialist through your SE. Learn about HP Consulting products and try to anticipate when you'll need these as you learn more about the 3000. This can be indispensable, and is also a good way to gain access to RPG specialists at HP. Finally, learn how to properly use PICS for RPG questions — an RPG specialist need not be on PICS for you to receive satisfactory response and resolution.

THIRD PARTY ASSISTANCE: When you need prolonged hand-holding and long-term help, there are sometimes third-party software suppliers who can provide help in RPG/3000 expertise. These are scarce, but nonetheless, have your sales representative check with your local third-party sales representative.

USER COMMUNITY — Locate all user groups who can provide a forum for discussing RPG-related topics, and provide a network you can call upon when necessary. HPGSUG, local RUGs, and a special interest group can all help, especially in providing you with an RPG toolbox. No special interest group currently exists. If you think it should, then help form one.

SELF TRAINING — This is probably the most important single difference between being an IBM GSD user and an HP3000 user. MPE is easily accessed by RPG users, and you can frequently solve your own problems by reading reference manuals, HPGSUG papers, etc. I have been very successful with this, and it allows you to share your solutions with others as you develop them. Again, talk to your SE to learn about all of the information that's already in your own installation.

2: Adopt MPE as a Design Determinant

Anything you do will at some point invoke MPE code. If you learn as much as possible about MPE and subsystems early, you will not be fighting with them later in debugging your RPG applications. As was stated before, this is the single most important key point in being successful with RPG/3000 — RPG calls all of the same intrinsics that COBOL, BASIC, etc. call. Here are the things you should master, along with suggested resources necessary to master them:

MPE INTRINSICS: Learn about the MPE Intrinsics — these are the basis for just about every function performed by the system. The MPE Intrinsics reference manual will provide enough information; there are sections related to Using the Intrinsics which contain good explanations of what they're useful for. Without a CALL verb, RPG can't do much with these directly, but this will still be very valuable knowledge in design and debugging. HPGSUG proceedings and HP consulting can help to solidify this knowledge.

FILE SYSTEM: Though it is actually part of the information in the Intrinsics manual, learn how the file system works. Your RPG code calls file system intrinsics for you, so you should know what you're asking MPE to do, as well as what it can do in general. Sepcifically, focus on FOPEN as it applies to RPG. This will

help you learn about the three biggest problems in RPG conversions: Buffering, Sharing, and Locking. If you understand how MPE does these things, it is much easier to ask RPG to do what you want. Again, HP consulting can be helpful here, as are issues 24 and 25 of the HP3000 Communicator.

SPECIAL CAPABILITIES: Again a subset of MPE intrinsics, two special capabilities can provide you with help in designing and converting on-line RPG systems. These are Multiple Rin (MR), which allows multiple concurrent

FLOCKS (and should be unnecessary in new systems), and Process Handling (PH), which allows your program to run another (son) program and suspend until it has completed execution.

STACK ARCHITECTURE: Learn what happens when you run a program, in terms of Code Segments, Data Stacks, Extra Data Stacks, what these terms mean in the first place (it's really very simple), and how they will affect you in the future. General reading and SE assistance will explain these things.

IMAGE: Though your converted systems will not employ IMAGE, the earlier you begin to use it, the more stable your environment will become. IMAGE is the most reliable and efficient data structure available on the 3000. Needless to say, the IMAGE course should be the first step you take, followed by RPG/IMAGE consulting, reading, and a small-scale project to let you become comfortable before you embark upon any significant new development project whcih will employ IMAGE. Converting old applications to IMAGE usually doesn't make sense, though it can be done easily and will improve your application stability.

3: Understand the Elements of Interactive Systems

Again, the choice of an HP3000 implies a change in the general approach you will be taking, and one of the most important differences is the interactive nature of the new systems you will be developing. When you offer a user an interactive system, you will need more protection against error, better recovery capability, and improved up-time. Technically, this re-alignment will require you to understand how to best use and control all peripheral equipment you will place in the hands of the user. This will involve your mastering two basic areas:

DEVICE CONTROL: Terminals and printers can be controlled directly through the use of a subset of MPE intrinsics, especially FCONTROL. Again, learn how you can control devices within the constraints of RPG. Many large systems isolate the user from MPE by using terminal monitoring and control programs which make it impossible for the user to get to a colon prompt. Though this is not possible with RPG, a terminal monitoring facility could launch RPG applications when a terminal response is requested. RPG allows you to read/write to $STDIN/$STDLIST; try all of the possible File specifications you could use to do this, and settle on one you're most comfortable with (I prefer to define an input demand and an output file). Finally, learn about escape sequences for terminal control, and all of the techniques you could use to send these to the terminal. This is easily done from RPG programs, though many RPG users are not aware of this capability.

VPLUS: Like IMAGE, a thorough understanding of VPLUS is essential to development of terminal-based RPG/3000 applications. This is probably the most controversial RPG interface, but you can be relatively successful in writing VPLUS/RPG code by following the same steps suggested earlier for IMAGE. If you try to learn RPG/VPLUS on your own and without the VPLUS class and SE consulting, chances are that you will be very frustrated, with unhappy users and unstable programs.

4: Re-Think Your RPG Design and Programming Techniques

Finally, once you've absorbed all of the material presented in the above pages, it is also beneficial to review the kind of programming guidelines you've used in the past. What you've learned about the possibilities of the HP3000 will allow you to be much more creative with your programs than you might have been in the past. Again, the following basic areas should be explored:

STRUCTURED DESIGN/PROTOTYPING: Programmers in other environments have been benefiting from two major design techniques, Structured Design and Prototyping. Do some general reading to familiarize yourself with these concepts, and determine whether either might not be of some benefit. Though interest in this technique seems to be waning, structured design does allow you to begin to start thinking in terms of small modular programs, an idea which MPE will allow you to employ easily. Modular applications allow you to develop your system as a tree of processes which you can develop, test and debug in a "top-down" fashion, which is far easier than traditional RPG development techniques. Secondly, prototyping is an idea which is becoming increasingly more prevalent because of the attendant low development costs associated with it. HP's RAPID products employ these techniques, and RPG shops who develop general programs and routines could also employ the same technique, though with not nearly the speed of development. Since RPG is a cryptic, table-driven language, it fits well with the idea of procedural brevity which is required in prototyping. Again, general reading and contact with user groups can help you learn more about these ideas.

CYCLE CONTROL: Because RPG/3000 is internally very different from IBM's RPGII, it is possible to use RPG similarly to other languages by eliminating automatic I/O (cycle driven files), and doing reading, writ-

ing, and calculations all within your calc. specifications. This is a heated argument elimination of automatic I/O does not mean you are in total control, but some users prefer this technique. Overhead in this case is not higher, as it is in IBM environments.

PROGRAM STRUCTURE: IBM indexes program efficiency to avoidance of high- overhead calculations. On the HP3000, the lowest overhead program is the program with the fewest statements and most logical calculation structure. If you use straightforward mainline code with nested subroutines, this will usually result in less object code. It will be important for you to learn about the RPG compiler internals and segmentation if this is important to you. Communicator #24 contains an interesting article related to RPG segmentation. Your SEGMENTER is the best tool you can employ to see what happens when you write a certain type of code. Be careful not to expect this to be as important as it was on IBM GSD equipment — all RPG/3000 code is not compiled, and MPE lets sloppy code execute quickly. . .

EXITING RPG: RPG doesn't take total advantage of MPE (neither does any other language); sometimes it makes sense to use the EXIT calculation to invoke a procedure written in another language. For instance, if you need to execute an MPE command from an RPG program, you could simply EXIT to a simple SPL routine which calls the COMMAND intrinsic (this could also be written in other languages), passing the command from your RPG program. This technique is almost indispensable in successful RPG/3000 systems. To learn more about this, look in the RPG reference manual, and get a copy of the REALRPG facility from the HPGSUG library, release 08. Very few HP3000 shops are monolingual.

CONCLUSION

Programming languages are simply vehicles to make computers Because of this, it is important to focus on the architecture and constructs used in the computer you're using to be successful in using a language make your computer "go fast." In the case of RPG users, we learned how to program without understanding what the programs were asking the computer to do, except in general terms. Hopefully, this paper will re-emphasize the importance of understanding the relationship of success to an understanding the HP3000 on its own terms. If the methodology outlined above is employed in an RPG/3000 installation, regardless of the age of the installation, I am quite certain that the user will be totally successful in developing high-quality interactive systems on the HP3000. As in any endeavor, attitude and organization will eventually determine how successful that endeavor is.

BIBLIOGRAPHY

1. Walmsley, David E., "RPG/3000 Programming Efficiency," HPGSUG Proceedings, September 1977, pp 42-48.
2. Todoroff, Gary, "RPG II: Report Writer of Programming Language," NOWRUG Presentation, May 1980.
3. King, David, "Current Methodologies in Structured Design," Computerworld, September, 1981, pp ID25-44.
4. Schulz, Duane, "Living in an RPG/3000 Environment," HPGSUG Proceedings, February, 1980, pp 2/75-83.
5. Schulz, Duane, Cummings, Randy, and Stevens, Brian, "RPG/3000 Application Development Course," HP SEO, Wilsonville, OR/King or Prussia, PA, December, 1981.

An Experimental, Comprehensive Data Dictionary

Thomas R. Harbron
Professor of Computer Science
Anderson College
Anderson, Indiana

Christopher M. Funk
President, C. M. Funk & Co.
Lafayette, Indiana

ABSTRACT

This paper describes an experimental Comprehensive Data Dictionary (CDD). The purpose of the CDD is to describe all data objects precisely, from bits to databases, so that programs may manipulate these objects without continually redefining them.

The most complex part of the description concerns the ways in which data objects relate to each other. By precisely describing these relationships, the CDD allows relatively simple processors to perform the functions of database management systems (IMAGE), screen drivers (V/3000), report generators, query processors (QUERY) and other subsystems.

Application programs may be developed with relatively little effort since all descriptions, relationships, and conversions are described by the CDD and need not be included in the program.

The experimental CDD is described in detail and the experience of mapping applications into it is shared. Strengths and weaknesses are assessed and the direction of future developments indicated.

INTRODUCTION

Centrality of Data

A mature view of data processing is that programs are functions operating on data. This idea may be expressed in mathematical notation as:

$$Y := F(X)$$

where Y is the set of output data, X is the set of input data and F is the function of the program.

Very often the function is fairly simple and, when the program is examined, one finds that most of the program is concerned with describing either the data in sets X and Y, or elementary transformations between them. The actual, functional parts of the program constitute a relatively small portion of the total code. The problem is compounded by the need to repeat the data descriptions and elementary transformations in each and every program.

It is the purpose of a Comprehensive Data Dictionary to provide these descriptions in one central location. This has three immediate benefits for programs. First it eliminates the need to repeat the descriptions in each program, thereby considerably shortening the programs. Second, it provides a single, consistent description for all programs, thus eliminating conflicts. Third, it makes it possible to build general-purpose programs such as query processors, report generators, etc., thereby eliminating the need for most programming.

Traditional Weakness of Data Descriptions

The problem may not have begun with FORTRAN, but as the first popular, high-level language, FORTRAN did much to promote the idea that code was the main problem of data processing and data was only incidental to the code. Early FORTRAN compilers not only didn't require data declarations but, except for arrays, did not even permit them. Variables were "declared" simply by mentioning their names in the program. Data type was determined by the first letter of the variable name.

Later languages such as COBOL, and most recently PASCAL, have done much to restore data descriptions to their proper position where data within the program is concerned. Likewise systems developed in the last decade have included descriptions of data external to programs such as the schema of IMAGE and forms file of V/3000.

Each of these data descriptions, however, has only spanned a small and specific portion of the data used by an application. Not only does this result in a fragmented description, but numerous problems are created when the various descriptions do not totally coincide at the boundaries between them.

The Comprehensive Data Dictionary

The purpose of the Comprehensive Data Dictionary (CDD) is to provide a single source for descriptions of all data elements in an application. This includes simple data items, aggregations such as arrays, records, internal files including databases, and external files including reports and screens. Although not properly part of the data descriptions, it is easy to add access and security information to the CDD as well.

It is important to implement the CDD in such a way that it can be easily read by an automatic processor

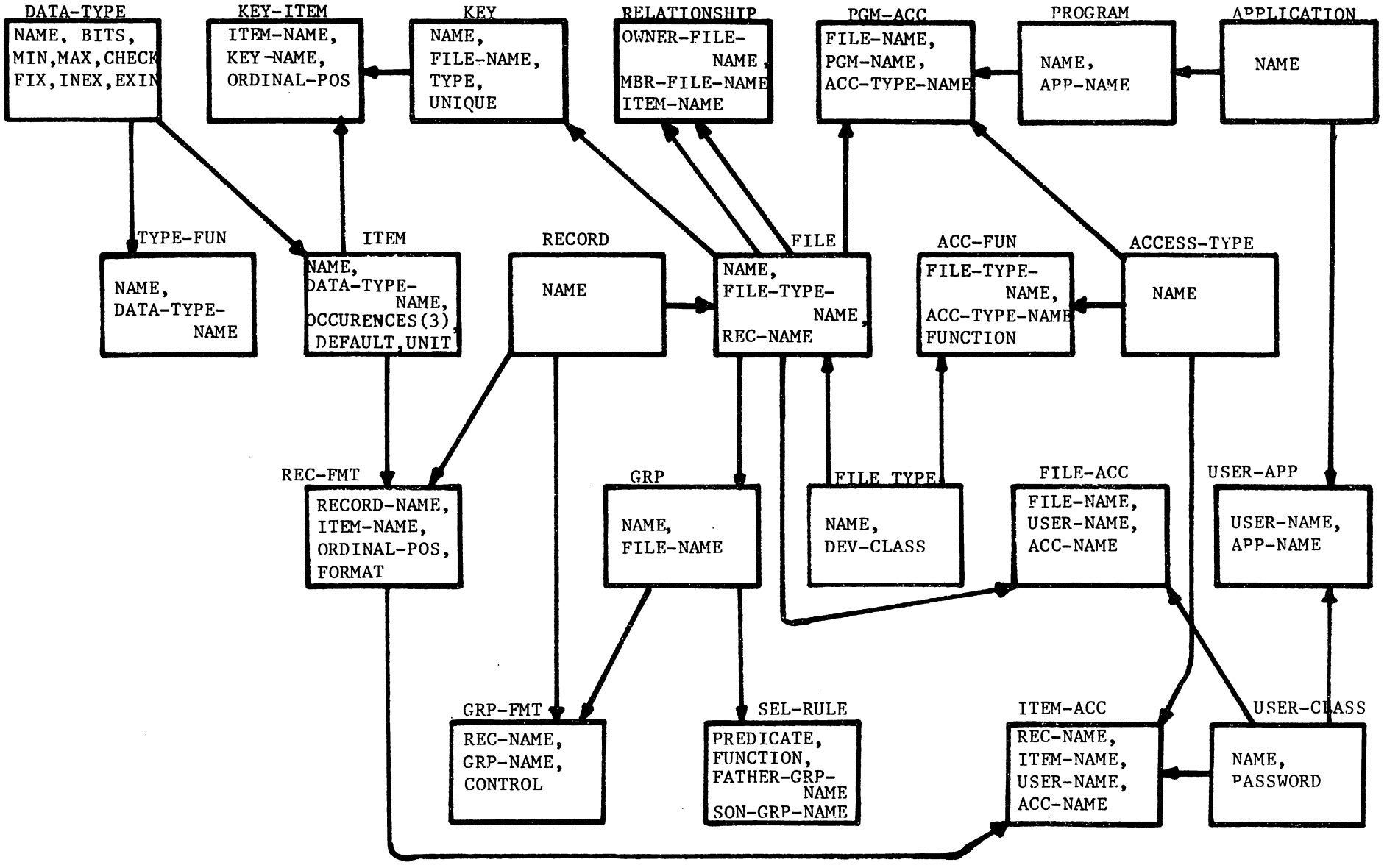

DATA-TYPE
NAME, BITS,
MIN,MAX,CHECK
FIX,INEX,EXIN
KEY-ITEM
ITEM-NAME,
KEY-NAME,
ORDINAL-POS
KEY
NAME,
FILE-NAME,
TYPE,
UNIQUE
RELATIONSHIP
OWNER-FILE-
NAME,
MBR-FILE-NAME
ITEM-NAME
PGM-ACC
FILE-NAME,
PGM-NAME,
ACC-TYPE-NAME
PROGRAM
NAME,
APP-NAME
APPLICATION
NAME
TYPE-FUN
NAME,
DATA-TYPE-
NAME
ITEM
NAME,
DATA-TYPE-
NAME,
OCCURENCES(3),
DEFAULT,UNIT
RECORD
NAME
FILE
NAME,
FILE-TYPE-
NAME,
REC-NAME
ACC-FUN
FILE-TYPE-
NAME,
ACC-TYPE-NAME
FUNCTION
ACCESS-TYPE
NAME
REC-FMT
RECORD-NAME,
ITEM-NAME,
ORDINAL-POS,
FORMAT
GRP
NAME,
FILE-NAME
FILE TYPE
NAME,
DEV-CLASS
FILE-ACC
FILE-NAME,
USER-NAME,
ACC-NAME
USER-APP
USER-NAME,
APP-NAME
GRP-FMT
REC-NAME,
GRP-NAME,
CONTROL
SEL-RULE
PREDICATE,
FUNCTION,
FATHER-GRP-
NAME
SON-GRP-NAME
ITEM-ACC
REC-NAME,
ITEM-NAME,
USER-NAME,
ACC-NAME
USER-CLASS
NAME,
PASSWORD

Figure 1

(report generator, query processor, program generator, etc.) as well as by people. Typically, the processor would read and store internally the descriptions relevant to the particular function being performed at the time.

The Experiment

There comes a point where theoretical work must be reconciled with the "real world." That is the purpose of this experiment. The CDD model has been derived on a solid theoretical basis. The model conforms to that of a normalized network database. It has been implemented using a relation database system.

The CDD, thus implemented, has been used to, first, describe itself, a non-trivial exercise. Next a variety of applications, drawn from a production environment, have been described in the CDD. Some weaknesses have been uncovered by this process, as well as some things that work very well.

THE COMPREHENSIVE DATA DICTIONARY

A data structure diagram is used to describe the CDD as shown in Figure 1. This model, with its 22 entities, 29 relationships, and 37 attributes, is too detailed to describe as a whole. Instead, it will be described in six parts in the following sections. The reader may, however, wish to refer to Figure 1 from time-to-time to see how the various parts are related.

Data-Item Part

This part of the CDD describes data items, their aggregations, and their components. This part of the CDD is shown in Figure 2.

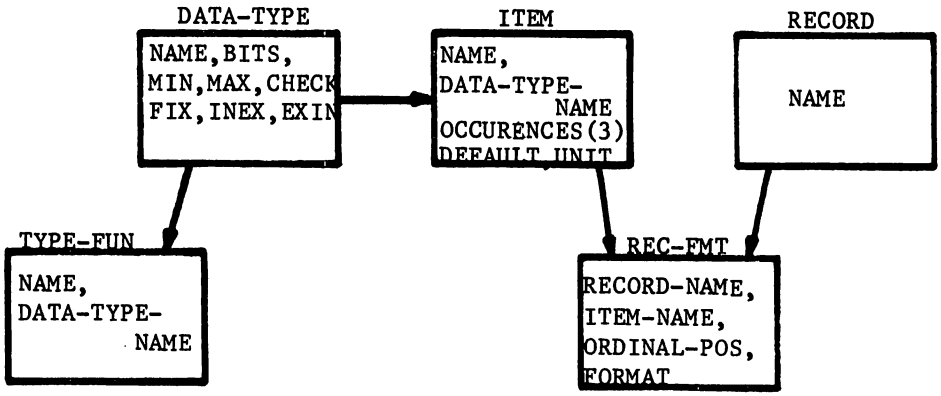


Figure 2

Before data items can be defined, it is necessary to define the basic data-types. Data-types may be defined in terms of their descriptions and the operations that may be performed on them. The descriptions and a basic set of functions are contained in the DATA-TYPE entity. Arithmetic, logical, and other functions are named, but not described in the TYPE-FUN entity.

An item may be a single occurrence of a date-type, or an array of up to three dimensions. A record is an aggregation of items and may be either an internal file, such as a disk file or database, or an external file such as a screen or report.

Record-format describes how items are related to records including position and format.

The following contains a description of each entity, its attributes, and relationships for this part.

ENTITY: DATA-TYPE

This entity describes a fundamental data-type such as byte, integer, real, etc. Only rarely should it be necessary to add a data-type once the basic set is in place. However, provision is made to describe new data-types in terms of their attributes. No semantic descriptions are provided.

Attribute: NAME

An ASCII character string of eight bytes containing the name of the data-type. This name will be referenced from other entities.

Attribute: BITS

The number of bits required by this data-type. Data-types will be assumed to start on word boundaries (high order end) except where assembled into arrays where they may be packed.

Attribute: MIN

This is the minimum value allowed for data of this type. Sixty-four bits are allowed for its representation. However, only the number of bits specified by the "BITS" attribute are used. If the numeric value of MIN cannot be represented in sixty-four bits or less, the value will be left justified and all truncated bits will be assumed to be zeroes.

Attribute: MAX

This is the maximum value allowed for data of this type. Storage is the same as for "MIN." If the numeric value of MAX cannot be represented in sixty-four bits

or less the value will be left justified and all truncated bits will be assumed to be zeroes.

Attribute: CHECK

This is the name of a procedure which will check representations of this data-type to see if they contain legal values. It returns only a true/false indication.

Attribute: FIX

This is the name of a procedure which will check representations of this data-type to see if they contain legal values. In case of an illegal value, it will replace the illegal value with a default value appropriate to the illegal value. It may also return an indication of the error.

Attribute: INEX

This is the name of a procedure which will convert an internal representation of this data-type to an external (ASCII) form. In addition to the value of the data-item, it may also use a format description (see REC-FMT) to specify options in the conversion.

Attribute: EXIN

This is the name of a procedure which will convert an external representation of this data-type to an internal form. Again, a format description may be used to specify options in the conversion.

Relationship: TYPE-FUN

DATA-TYPE is related 1:N to TYPE-FUN. Each related TYPE-FUN is a legitimate function to use with this DATA-TYPE. The linking data-item is DATA-TYPE-NAME.

Relationship: ITEM

DATA-TYPE is related 1:N to ITEM. Each related ITEM is of this DATA-TYPE. The linking data-item is DATA-TYPE-NAME.

ENTITY: TYPE-FUN

This entity represents each function that is associated with a data-type.

Attribute: NAME

An ASCII character string of eight bytes that gives the name of the function.

Attribute: DATA-TYPE-NAME

The name of the data-type for which this is a function.

Relationship: DATA-TYPE

DATA-FUN is related N:1 to DATA-TYPE. The linking data-item is DATA-TYPE-NAME.

ENTITY: ITEM

This entity describes each unique data-item. The item may be a simple variable, or an array in 1, 2, or 3 dimensions.

Attribute: NAME

An ASCII character string of 12 bytes containing the name of the item.

Attribute: DATA-TYPE-NAME

The data-type of which this item is one occurrence.

Attribute: DEFAULT

A default value which is to be used for this item when no other value is available. Sixty-four bits are allowed for its representation, but only the bits required are used. In the case of array items, only the value for one element of the array is given.

Attribute: OCCURRENCES

This is a triple valued attribute which gives the three dimensions of the array if this item is an array. For a simple data-item, this attribute will have the value 1,1,1. For a one-dimensional array of order N, it will have the values N,1,1. For a two-dimensional array, values M,N,1; for three dimensions, values L,M,N.

Attribute: UNIT

This attribute is an ASCII string of eight characters used to indicate the unit of measurement, such as feet, yards, meters, etc. if no units of measurement are required, this field will be null.

Relationship: DATA-TYPE

ITEM is related N:1 to DATA-TYPE. Each item is of exactly one DATA-TYPE. DATA-TYPE-NAME is the linking data-item.

Relationship: REC-FMT

ITEM is related 1:N to REC-FMT. The linking data-item is ITEM-NAME.

Relationship: KEY-ITEM

ITEM is related 1:N to KEY-ITEM, with ITEM-NAME as the linking data-item. This relationship indicates which items are used as keys.

ENTITY: RECORD

This entity names a logical record which can be a part of one or more files. The record contains one or more data-items and may be of internal or external value.

Attribute: NAME

An ASCII character string sixteen bytes long containing the name of the record. This name will be referenced by other entities.

Relationship: REC-FMT

RECORD is related 1:N to REC-FMT, with RECORD-NAME as the linking data-item. This relationship defines the items contained in the record, their location, and their format.

Relationship: FILE

RECORD is related 1:N to FILE, and the linking data-item is RECORD-NAME. This relationship exists only for internal files and identifies the files in which each record occurs.

Relationship: GRP-FMT

RECORD is related 1:N to GRP-FMT, with the linking data-item being RECORD-NAME. This relationship exists only for external files and identifies the groups (and ultimately files) in which each record occurs.

ENTITY: REC-FMT

This entity (record format) represents the unique in-

tersection of one item and one record. The entity contains information on how the item is related to the record.

Attribute: RECORD-NAME

The record name of which REC-FMT is a member.

Attribute: ITEM-NAME

The name of the item being described.

Attribute: ORDINAL-POS

An integer stating the ordinal position (1st, 2nd, 3rd, etc.) of the item in the record.

Attribute: FORMAT

This is a description of the format of the item for this particular record. This attribute will be used to determine dollar signs, commas, and other external features. The internal representation is indicated by a default format.

Relationship: ITEM-ACC

REC-FMT is related 1:N to ITEM-ACC, and the ITEM-NAME provides the link. This relationship exists as part of the security provisions and determines the access allowed each user-class to each item within each record.

Relationship: ITEM

REC-FMT is related N:1 to ITEM. The linking data-item is ITEM-NAME.

Relationship: RECORD

REC-FMT is related N:1 to RECORD, with RECORD-NAME providing the linkage.

* * *

Internal File Part

This portion of the CDD describes internal files including disk files, databases, etc. This part of the CDD is shown in Figure 3.

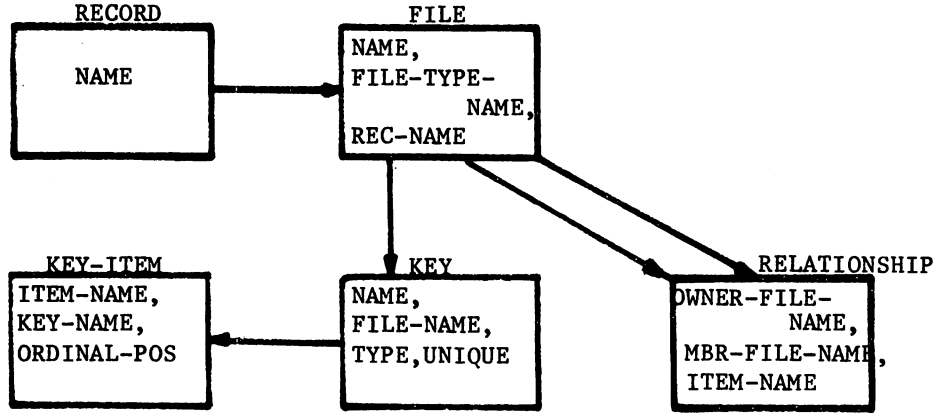


Figure 3

Each record type occurs in one or more files. Each file, usually has one or more keys by which records may be identified and retrieved. Each key, in turn, may consist of one or more data-items. The relationship between keys and data items is described by the entity KEY-ITEM.

The entity RELATIONSHIP is used to describe the relationship between records in one file and records in another file. For a given relationship, a file is either the owner or a member of the relationship. If a file is the owner of a relationship, the following conditions prevail:

1. Each owner record is related to zero or more records in the member file.

2. Each owner record shares with its member records a common value of the linking data-item.

3. An owner record may not be deleted if it is related to one or more member records.

The reader may recognize IMAGE "master" records as being owner types. In IMAGE the relationships are indicated by "chains" of pointers. Likewise, from the following constraints on member records, it may be seen that IMAGE "detail" records are member records.

1. Each member record is related to exactly one owner record in the relationship.

2. All member records share with their owner record a common value of the linking data-item.

3. A member record may not be added if no owner record exists with which it shares a common value of the linking data-item.

These rules not only define how a relationship is established between records in different files, but also prevent the infamous insertion and deletion anamolies from occurring in a normalized database. A file may simultaneously be a member of zero or more relationships and the owner of zero or more relationships. Note that in data structure diagrams, such as Figure 1, the arrow always points from the owner to the member in a relationship.

This description of internal files with keys and relationships, is equivalent to a database schema. Thus the

CDD subsumes the part of the database management system.

The entities not previously described are as follows:

ENTITY: FILE

The entity FILE describes a unique file of a given name. Files can be external in form, such as reports and screens, or internal in the form of disk and other storage medium files. External files may contain a variety of records and these records are collected into groups. The entities GRP and GRP-FMT are used to relate records to external files. Internal files normally contain one type of record. This relationship is shown by the 1:N relationship from record to file. An internal file may have one or more keys and relationships between internal files are given by the RELATIONSHIP entity.

Attribute: NAME

An ASCII character string with a maximum of twenty-six bytes containing the file name, group name, and account name necessary for accessing the file. This name will be referenced by other entities.

Attribute: FILE-TYPE-NAME

The name of the file-type to which a given file belongs.

Attribute: RECORD-NAME

The name of the record which occurs repeatedly to form the file. This attribute is valid only for internal files and will default when the file is of external form.

Relationship: FILE-TYPE

FILE is related N:1 to FILE-TYPE. The linking data-item is FILE-TYPE-NAME. This relationship indicates the file-type and, by implication, the functions for each file.

Relationship: FILE-ACC

This is a 1:N relationship between FILE and FILE-ACC with a linking data-item of FILE-NAME. The relationship indicates the access modes allowed to specific user-class for this file.

Relationship: PGM-ACC

FILE is related 1:N to PGM-ACC. The linking data-item is FILE-NAME. This relationship indicates the access mode used by a given program for each file.

Relationship: RECORD

FILE is related N:1 to RECORD with the linking data-item being RECORD-NAME. This relationship is valid only for files of an internal form and shows the normal pattern of one record type for an internal file.

Relationship: KEY

FILE is related 1:N to KEY and the linking data-item is FILE-NAME. Entity KEY and this relationship are valid only for internal files. Each key is a legitimate search item for the related file.

Relationship: GROUP

FILE is related 1:N to GROUP with the linking data-item being FILE-NAME. This relationship is valid only for external files and indicates the groups of records that are included in this file.

Relationship: OWNER-RELATIONSHIP

FILE is related to the entity RELATIONSHIP on the order of 1:N with OWNER-FILE-NAME being the linking data-item. This links each owner file to its corresponding relationships.

Relationship: MEMBER-RELATIONSHIP

FILE is related to the entity RELATIONSHIP on the order of 1:N with MEMBER-FILE-NAME being the linking data item. This links each member file to its corresponding relationship.

ENTITY: RELATIONSHIP

Attribute: OWNER-FILE-NAME

An ASCII string of 26 bytes that names the file which "owns" the relationship.

Attribute: MEMBER-FILE-NAME

An ASCII string of 12 bytes that names the file which is a "member" of the relationship.

Attribute: ITEM-NAME

An ASCII string of 12 bytes that names the data-item whose value is shared by the owner record and member records in this relationship.

Relationship: OWNER-FILE

RELATIONSHIP is related N:1 to FILE with OWNER-FILE-NAME being the linking data item. This links each member file to its corresponding relationships.

Relationship: MEMBER-FILE

RELATIONSHIP is related N:1 to FILE with MEMBER-FILE-NAME being the linking data-item. This links each member file to its corresponding relationships.

ENTITY: KEY

This entity identifies any and all keys for each internal file. The entity contains information on the name of the key, the file name to which it belongs, and the type of key.

Attribute: NAME

An ASCII string of 16 bytes containing the name of the key. This name will be referenced by KEY-ITEM.

Attribute: FILE-NAME

The name of the file to which a given key belongs.

Attribute: TYPE

This attribute is used to define the method of accessing a record by using the key. The type will differ according to whether the file is a sequential file, database file, etc.

Attribute: UNIQUE

This attribute has a value which is either true or false. If true, then each value of the key must be distinct from all other values of the key.

Relationship: KEY-ITEM

KEY is related 1:N to KEY-ITEM, with KEY-NAME providing the linkage. Any given key consists of one or more occurrences of KEY-ITEM. This allows a key to consist of composite data-items.

Relationship: FILE

KEY is related N:1 to FILE, with FILE-NAME providing the linkage.

ENTITY: KEY-ITEM

This entity represents the unique intersection of one key and one item. The entity contains information on how the item is related to the key.

Attribute: ITEM-NAME

The name of the item being described.

Attribute: KEY-NAME

The key name of which KEY-ITEM is a member.

Attribute: ORDINAL-POS

An integer stating the ordinal position (1st, 2nd, 3rd, etc.) of the item in the key.

Relationship: KEY

KEY-ITEM is related N:1 to KEY, with KEY-NAME being the linking data-item.

Relationship: ITEM

KEY-ITEM is related N:1 to ITEM, with ITEM-NAME being the linking data-item.

* * *

External File Part

External files are those which are displayed externally from the computer system and generally are intended to be read and/or written by people as well as machines. Included in this category are formatted screens, reports, and graphical presentations.

Unlike internal files, which normally contain only one type of record, external files typically contain a variety of records. Organizing and sequencing this variety of records is the principal challenge in this part. The entities concerned in this organization are shown in Figure 4.

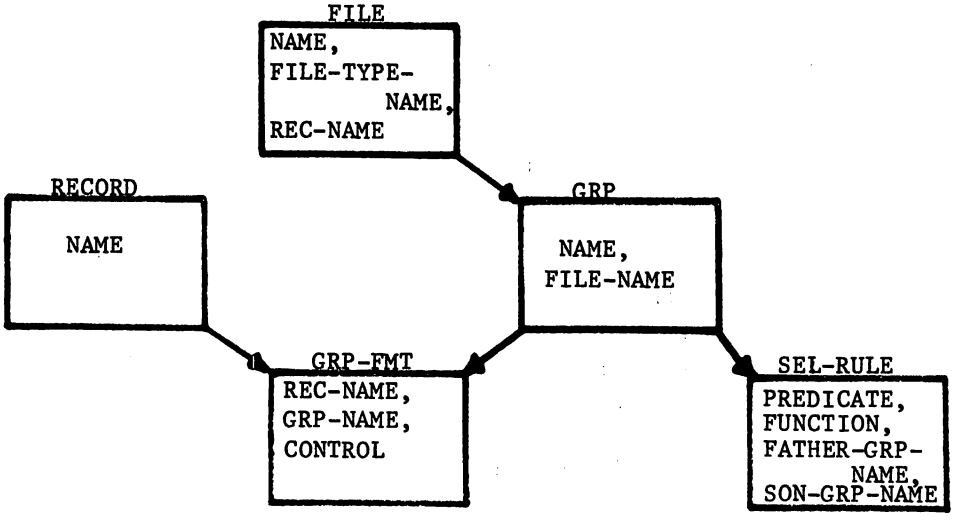


Figure 4

Each file consists of an aggregation of "groups" (GRP). A group is a group of records. The placement of each record within the group is controlled by the entity "group-format" (GRP-FMT). Since, typically, the rules for determining which group follows the previous one are data dependent, provision is made for a "selection rule" (SEL-RULE) to determine the sequence of groups within the file.

Descriptions of the entities from this part are as follows:

ENTITY: GROUP

This entity exists for external files only and names each specific group of records which are part of a given file. An external file consists of one or more groups, each group containing one or more records.

Attribute: NAME

An ASCII character string of 16 bytes that names each group.

Attribute: FILE-NAME

The name of the file to which the group belongs.

Relationship: GRP-FMT

GROUP is related 1:N to GRP-FMT, with GROUP-NAME providing the link. Any given group consists of one or more occurrences of GRP-FMT. This relationship defines the records contained in the group.

Relationship: SEL-RULE

GROUP is related 1:N to SEL-RULE, with the link-

ing data-item being GROUP-NAME. SEL-RULE (selection rule) determines if the current group will be repeated or a new group will be selected.

Relationship: FILE

GROUP is related N:1 to FILE, with the linking data-item being FILE-NAME.

ENTITY: GRP-FMT

This entity (group format) represents the unique intersection of one record and one group. It contains information on how the record is related to the group.

Attribute: RECORD-NAME

The name of the record being described.

Attribute: GROUP-NAME

The group name of which GRP-FMT is a member.

Attribute: CONTROL

An ASCII string of eight bytes used to indicate the placement of the record within the group.

Relationship: RECORD

GRP-FMT is related N:1 to RECORD, with RECORD-NAME being the linking data-item.

Relationship: GROUP

GRP-FMT is related N:1 to GROUP, with GROUP-NAME being the linking data-item.

ENTITY: SEL-RULE

This entity (selection rule) is used to determine if the current group will be repeated, a new group will be selected, or the file terminated. The entity contains information on which group is to be selected and which function to use (append, replace, add, etc.).

Attribute: PREDICATE

An ASCII string of 28 characters which is tested to determine which rule will be selected. The following conditions prevail:

1. Each predicate is a proposition which is either true or false when tested.

2. The predicates are tested in the order given, and the first predicate found true prevails. Subsequent predicates are not tested.

3. Each predicate consists of a data-item name, an operator, and either a constant or another data-item name.

4. Data-items must be described in the CCD. All constants and variables must be of the same data-type. Operators are >, =, <, >=, <=, <>.

Attribute: FUNCTION

An ASCII string of eight characters containing the function to be used. The following functions are available:

REPEATA	Repeat, appended; this option repeats the current group and appends it to the previous group.
REPEATO	Repeat, overlayed; this option repeats the current group and overlays the previous group (this option is designed for use with screens).
NEXTA	Next group, appended; this function obtains the next group and appends it to the previous group.
NEXTC	Next group, cleared; this function obtains the next group and will clear the screen (or go to the top of the next page) before displaying the group.
TERMINATE	End of file; no new groups are obtained.

Attribute: FATHER-GROUP-NAME

The GROUP-NAME of the father of the current group. This attribute is used when the rule references the previous group.

Attribute: SON-GROUP-NAME

The GROUP-NAME of the son of the current group. This attribute is used when the rule references the next group.

Relationship: GROUP

SEL-RULE is related N:1 to GROUP, with the GROUP-NAME providing the linkage. The GROUP-NAME can be either the father of the current group or the son of the current group.

* * *

Access Part

Like data-items, a complete description of files must include the functions that operate upon them. These are the access functions which this section is concerned with. The relevant entities are shown in Figure 5.

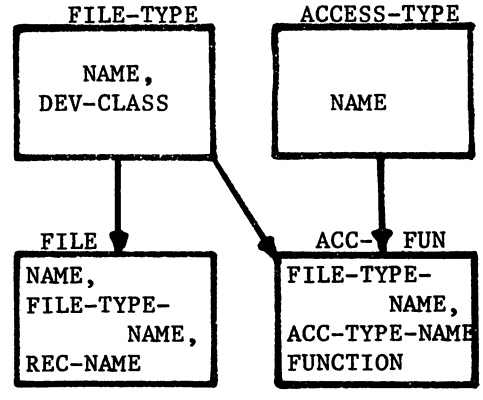


Figure 5

Each file must be of a type described by FILE-TYPE. These types may include sequential, direct access (hashing), indexed (KSAM, RELATE), IMAGE or other files. Each file contains an attribute which links it to a previously defined file type.

Likewise, there is a set of generic functions for files including read only, append only, update, read/write, etc. These are described in ACCESS-TYPE.

For each file-type and access-type, there is usually one function which provides that mode of access for

that particular file type. Not all file-types support all modes of access.

The descriptions of these entities are as follows:

ENTITY: FILE-TYPE

This entity specifies the type of each file, and by relationship, the access function for each file type.

Attribute: NAME

An ASCII character string of eight bytes used to name the various file types.

Attribute: DEV-CLASS

An ASCII character string of eight bytes which contains the device class name on which the file type resides.

Relationship: FILE

FILE-TYPE is related 1:N to FILE, with FILE-TYPE-NAME being the linking data-item. This relationship links all files of a given type.

Relationship: ACC-FUN

FILE-TYPE is related 1:N to ACC-FUN, with FILE-TYPE-NAME being the linking data-item. This relationship indicates the functions for access of a given file-type.

ENTITY: ACCESS-TYPE

This entity represents the various access modes that are available for items, files, and programs. In the attribute ACCESS-TYPE, each bit of the integer represents an access function. If the bit corresponding to a given function is set to 1 then that function is allowed in the access type. An access type can consist of one or more functions. The functions — and their corresponding bit positions — available as part of the dictionary are:

Bit	Function	Explanation
7	Exclusive	Access to data is given to this user only
8	Read	User is allowed to read data
9	Append	User may append new data
10	Update	User may modify existing data
11	Delete	User may delete records
12	Create	User may create files
13	Purge	User may delete files
14	Execute	User is allowed to execute or stream files
15	Locking	Files or items may be locked to prevent concurrent access

Examples are shown below.

Access Type	Bit Pattern	Decimal Value
Read only shared access	0000000010000000	128
Read, update shared access with locking	0000000010100001	161
Read, append, update exclusive access	0000000111100000	480

Attribute: NAME

An integer containing the bit code representing the corresponding access type. NAME is referenced from other entities.

Relationship: ACC-FUN

This is a 1:N relationship between ACCESS-TYPE and ACC-FUN which indicates the functions which are used for data manipulation when a particular access mode is prevalent. The linking data-item for this relationship is ACCESS-TYPE-NAME.

Relationship: PGM-ACC

ACCESS-TYPE is related 1:N to PGM-ACC with the linking data-item being ACCESS-TYPE-NAME. This relationship indicates the mode of access used by a given program to a given file.

Relationship: FILE-ACC

The entity ACCESS-TYPE is related 1:N with FILE-ACC and has a linking data-item of ACCESS-TYPE-NAME. This relationship indicates the files which are accessible by a given user.

Relationship: ITEM-ACC

The entity ACCESS-TYPE has a 1:N relationship to ITEM-ACC which represents the items which are accessible by a particular user. The linking data-item is ACCESS-TYPE-NAME.

ENTITY: ACC-FUN

This entity represents the function that is used with a given access mode to reference a certain file type. Functions are external to the Data Dictionary and will be referenced when a file access is requested.

Attribute: FILE-TYPE-NAME

The name of a file type for which a function is used.

Attribute: ACC-TYPE-NAME

The name of an access type for which a function is used.

Attribute: FUNCTION

The ASCII character string of eight bytes which names the function.

Relationship: ACCESS-TYPE

ACC-FUN is related N:1 to ACCESS-TYPE with a linking data-item of ACCESS-TYPE-NAME. This relationship indicates the functions which are used by a given access type.

Relationship: FILE-TYPE

This is an N:1 relationship between ACC-FUN and FILE-TYPE which indicates the functions that are used by a given file type. The linking data-item is FILE-TYPE-NAME.

* * *

Application Part

Although not properly a part of the data descriptions, it is helpful to have information on programs and applications in the CDD. Particularly useful is knowledge of

the relationships between programs and files; which programs use which files and in which mode of access.

This information is stored in the application part of the CDD as shown in Figure 6.

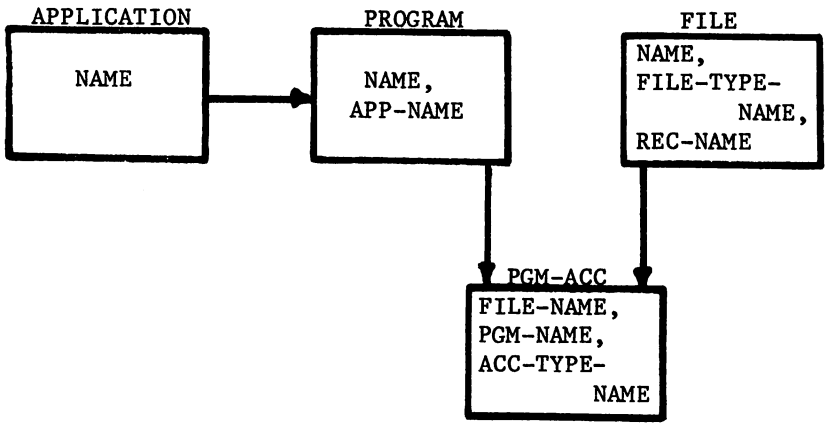


Figure 6

Each application area is given a name which is recorded in the entity APPLICATION. Each application owns a set of programs which are named in the PROGRAM entity. For each file accessed by each program, there is an entry in PGM-ACC which shows the mode of access for that particular program-file pair.

Since files commonly bridge application boundaries, there is no attempt to assign files to applications. The linkage exists implicitly through the programs.

The application part entities are described as follows:

ENTITY: APPLICATION

The entity APPLICATION represents the various applications whose data is described by the Data Dictionary. The users allowed to access an application are shown by the relationship to USER-APP.

Attribute: NAME

An ASCII character string of eight bytes containing the name of an application. This name will be referenced from other entities.

Relationship: PROGRAM

APPLICATION is related 1:N to PROGRAM with a linking data-item of APPLICATION-NAME. This relationship indicates the programs included in an application area.

Relationship: USER-APP

This is a 1:N relationship between APPLICATION and USER-APP which indicates the user's given access to an application. The linking data-item is APP-NAME.

ENTITY: PROGRAM

This entity gives the name of each program which is currently part of the Comprehensive Data Dictionary. The entity will also indicate the relationship any program has to an application area. The relationship between PROGRAM and PGM-ACC shows the access the program has to files.

Attribute: NAME

The ASCII character string of a maximum 26 bytes which contains the program name, group name, and account name necessary for accessing the file. This name will be referenced by other entities.

Attribute: APP-NAME

The name of the application to which this program belongs.

Relationship: APPLICATION

PROGRAM is related N:1 to APPLICATION. The linking data-item is APPLICATION-NAME. This relationship indicates the application area to which a program belongs.

Relationship: PGM-ACC

This is a 1:N relationship between PROGRAM and PGM-ACC which indicates the various access allowed between files and programs. The linking data-item is PROGRAM-NAME.

ENTITY: PGM-ACC

This entity is the unique intersection between ACCESS-TYPE, FILE, and PROGRAM. The entity represents the allowed file accesses for a given program. This entity is used to determine the mode of access allowed by each program to each file.

Attribute: FILE-NAME

The name of the file being accessed.

Attribute: PROGRAM-NAME

The program name of the program accessing the file.

Attribute: ACCESS-TYPE-NAME

The access type name which indicates the access mode for the access being defined.

Relationship: PROGRAM

PGM-ACC is related N:1 to PROGRAM with a linking data-item of PROGRAM-NAME. This relationship indicates which program is given access to the given file.

Relationship: FILE

This is a N:1 relationship between PGM-ACC and FILE which indicates the file which can be accessed by the program. The linking data-item is FILE-NAME.

Relationship: ACCESS-TYPE

The entity PGM-ACC is related N:1 to ACCESS-TYPE with a linking data-item of ACCESS-TYPE-NAME. This relationship indicates the type of access the program may use when referencing the file for a given PGM-ACC.

* * *

Security Part

As with the application part, security is not properly a part of the data description. However, it is a necessary part of any application using the CDD and may conveniently be accommodated here. This information is contained in the part of the CDD shown in Figure 7.

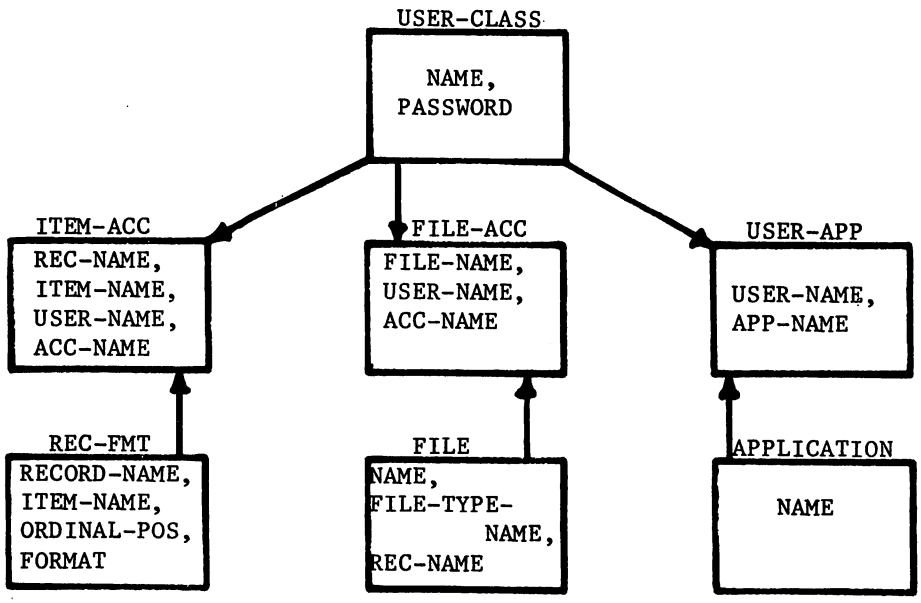


Figure 7

Users of the CDD, or applications described therein, are identified by their USER-CLASS-NAME. Each such name has a password associated with it to verify authenticity. The name and password are recorded in the USER-CLASS entity.

The applications, and hence programs, to which a given user-class has access are determined by entries in the USER-APP entity. An entry must occur here for each user-class/application pair that is allowed.

Data access is controlled at two levels. A user must be allowed access at both levels to be successful. Where a conflict exists between the levels, the most restrictive case prevails.

Access to files is controlled by the FILE-ACC entity. For each allowed user-class/file pair, an entry names the user-class, file, and acces mode allowed.

Access to individual data-items is controlled within the context of records. For example, a certain user-class may be allowed to read the data-item EMPLOYEE-NAME within the context of a production record but not allowed to see the same item in the context of a payroll record.

Data-item access is controlled by the ITEM-ACC entity. The item, record, user, and access mode are identified to allow the user access to the specified item within the specified record.

The security part entities are as follows:

ENTITY: USER-CLASS

This entity represents the different classes of users that will be able to reference the data described by the Data Dictionary. Each user class is allowed access to a limited set of applications, files, and data-items. The type of access is controlled in each case.

Attribute: NAME

An ASCII string of eight bytes containing the name of a user classification. This name will be referenced from other entities.

Attribute: PASSWORD

An ASCII string of eight bytes containing the password which controls the availability of specific user classification accesses.

Relationship: USER-APP

The entity USER-CLASS is related 1:N to USER-APP with a linking data-item of USER-CLASS-NAME. This relationship indicates which applications are accessible by a user classification, and the modes of access allowed.

Relationship: FILE-ACC

USER-CLASS is related 1:N to FILE-ACC. The linking data-item is USER-CLASS-NAME. The files accessible by a user classification and the mode of access are indicated through this relationship.

Relationship: ITEM-ACC

USER-CLASS is related 1:N to ITEM-ACC and the linking data-item is USER-CLASS-NAME. The items accessible by a user classification and the mode of access are indicated through this relationship.

ENTITY: USER-APP

This entity represents the unique intersection between a user classification and an application. The intersection shows a user classification that is allowed access to a given application.

Attribute: USER-CLASS-NAME

The name of a user classification for which an application access is being defined.

Attribute: APP-NAME

The name of an application for which an access is being defined.

Relationship: USER-CLASS

This is a N:1 relationship between USER-APP and USER-CLASS. The linking data-item is USER-CLASS-NAME. The relationship indicates the user classification which is given access to an application.

Relationship: APPLICATION

USER-APP is related N:1 to APPLICATION with the linking data-item of APPLICATION-NAME. The relationship indicates the application to which a user classification is given access.

ENTITY: ITEM-ACC

This entity represents the unique intersection of three entities REC-FMT, USER-CLASS, and ACCESS-TYPE. The intersection defines an access by indicating which user classification can reference an item in a particular record and what mode of access is permitted.

Attribute: ITEM-NAME

The name of the item for which an access is being defined.

Attribute: RECORD-NAME

The name of the record for which an access is being defined.

Attribute: USER-NAME

The name of the user classification for the access being defined.

Attribute: ACCESS-TYPE-NAME

The name of the access type or mode for the item access being defined.

Relationship: USER-CLASS

ITEM-ACC is related N:1 to USER-CLASS with the linking data-item being USER-CLASS-NAME. This relationship indicates the user classification that is given access to an item.

Relationship: ACCESS-TYPE

This N:1 relationship between ITEM-ACCESS and ACCESS-TYPE indicates the access mode allowed in referencing the item. The linking data-item is ACCESS-TYPE-NAME.

Relationship: REC-FMT

ITEM-ACCESS is related N:1 to REC-FMT and the relationship indicates which item of a particular record will be referenced through the access defined. The linking data-items for this relationship are RECORD-NAME and ITEM-NAME.

ENTITY: FILE-ACC

This entity is the unique intersection of USER-CLASS, ACCESS-TYPE and FILE. This entity represents the modes of access allowed in referencing a given file by a user class. The access is defined by the different relationships that are present in this entity.

Attribute: FILE-NAME

The name of the file for which the access is being defined.

Attribute: USER-CLASS-NAME

The name of the user classification for which the access is being defined.

Attribute: ACCESS-TYPE-NAME

The access type name which defines the file access.

Relationship: FILE

This is a N:1 relationship between FILE-ACC and FILE. The linking data-item is FILE-NAME. This relationship indicates the file which will be referenced through the access defined.

Relationship: USER-CLASS

FILE-ACC is related N:1 to USER-CLASS and the relationship indicates the user classification that is given access to a file. The linking data-item is USER-CLASS-NAME.

Relationship: ACCESS-TYPE

FILE-ACCESS is related N:1 to ACCESS-TYPE with a linking data-item of ACCESS-TYPE. The access mode allowed by the defined access is indicated by this relationship.

* * *

IMPLEMENTATION

Implementation of the CDD, done to date, is in three phases. First, a database is built to hold the data of the CDD. Second, the CDD is used to describe itself — a non-trivial exercise. Third, the CDD is used to describe some real-world applications.

The strengths and weaknesses of this CDD are assessed, based on the limited experience gained to date. Finally, future developments are briefly discussed.

Mapping the Model to a Database

The model of the CDD, described in the previous section, is in the form of a normalized, network, database. Thus, it is only natural to seek a database management system (DBMS) with which to implement it. Although IMAGE is based on the network model, it was rejected because of its rigidity and the limits of its two-level structure. Instead, RELATE/3000,* a relational DBMS was selected.

In mapping the model into RELATE, each entity becomes a Relation, or file. These files may each be indexed on any combination of keys. Attributes become data-items. Relationships cannot be explicitly shown in a relational DBMS, but are implicitly linked by shared data-item values.

Mapping the CDD into Itself

As a first exercise in mapping applications into the CDD, it was decided to map it into itself; i.e., use the CDD to describe itself. Since the CDD contains 22 entities, 37 data-items, and 29 relationships, the exercise is not trivial.

The initial mapping of the CDD into itself, using RELATE, is shown in Appendix A. Notice that some files are empty because the corresponding entities are not needed in this application. For example, at this time, there are no external files associated with the CDD, so the corresponding entities are empty.

Several small problems were encountered in this exercise. Several of the data-item and entity names had to be modified to conform to the naming conventions of RELATE. Since a full set of functions was not immediately defined for the standard data types, the TYPE-FUN entity was left empty. Likewise no programs were intially associated with the CDD.

Some problems of greater significance also appeared. One, that will doubtless reoccur in other applications, is that of composite data-items used as links in entity relationships. For example, both record-name and item-name are used as the linking item between REC-FMT and ITEM-ACC. Neither alone is sufficient. Yet provision is made for only one linking item in the RELATIONSHIP entity.

Another is the magnitude of records that can occur in some entities. For example, ITEM-ACC is limited only by the product of the number of records in ITEM, RECORD, ACCESS-TYPE, and USER-CLASS. At one time the number of records in these entities were 37, 22, 13, and 2 respectively giving a potential of 21,164 records in ITEM-ACC. While the actual number was only 284 it is still too large. Some kind of "wild-card" notation is being considered to reduce the number of records.

Another troublesome area is the representation of the values of MIN and MAX in DATA-TYPE, and DEFAULT in ITEM. The intention is that the binary or internal representations of these values be stored. However, this would require that these items be of different data-types in different records — a complexity beyond the ability of most DBMSs to handle. Two alternatives are apparent; either store them in external form, in which case all are stored as ASCII character strings; or declare them type long and left justify the actual value within the 64 bits.

Mapping Applications to the CDD

The press deadline for submission of this paper occurred too soon to allow much experimentation with real applications. The authors will be able to share these experiences when the paper is presented.

However, on the basis of early work done, some things have become obvious, and several changes or redefinitions are clearly indicated.

First, there is a substantial weakness in the area of composite data types. An additional entity needs to be created to link a composite data-type with its components. This will have several advantages over the present mechanism.

1. Arrays of any number of dimensions can be declared.
2. Composite types may have components of several different types.
3. Composite types may become components of more complex types.

Second, the whole access area is proving troublesome. Several issues need to be better defined including:

1. Better definitions of access modes and allowed combinations of modes.
2. A notation for item access that does not require a separate entry for each user-record-item intersection.

Third, some minor changes are needed in GROUP and GRP-FMT to accommodate the structure clash between external files and physical pages and screens. An attribute (LINE-NBR) can be added to GROUP to indicate the last line on which that group is allowed to begin. A current line number greater than this will trigger a new page.

Likewise a standard group must be added to each external file which will be inserted whenever a new page is triggered. This same group will also, automatically, begin each external file, thus eliminating initializing problems.

On the whole, real applications appear to be mapping in with very few other problems. In particular, the group structure for external file descriptions seems to work well. A final judgment must, however, await trials with "strange" external files as well as more standard ones.

---

*RELATE is a trademark of Computer Resources Incorporated, 2570 El Camino Real, Mountain View, CA 94040.

Future Developments

The next step is to complete the current development phase, i.e., testing the model against a variety of applications and refining it as indicated.

The next phase is to develop a "front-end" program to interface between the CDD and its manager. This program would perform the functions of adding, deleting, and modifying the contents of the CDD while checking for consistency. It would also provide formatted reports on the contents of the CDD.

To this point, the CDD will not have been used by processors to do production data processing. While it may prove very useful for documentation purposes, the principal value of the CDD is in its use in production. The development of the processors required to apply the CDD to production can proceed in three phases. While there is some overlap and interaction, they may proceed somewhat independently.

The first processor is a query/report/screen processor. It will move data between internal and external files. Thus, to produce a new report, it is only necessary to describe the report in the CDD. The processor can then produce the report from internal files. Likewise data could be transmitted between screens and internal files.

The second processor integrates the DBMS with the CDD. As mentioned earlier, presently available DBMSs each have a separate "schema" which describes only the data in the database. This processor combines the DBMS with the CDD so that internal files are described in only one place.

The third processor is a program generator which relies on the CDD for all data descriptions. This may either be a compiler or interpreter. In either case very high-level statements would allow most programs to be expressed in a fraction of the number of statements required by typical languages. By removing the data descriptions and conversions from the program, only the functional parts need be expressed.

CONCLUSION

The CDD has, initially, shown the capacity to contain the total data descriptions needed for applications. Thus, it is a suitable base on which to build sophisticated processors which will greatly reduce the need for applications programming.

Research will continue in this direction. Meanwhile, it is hoped that others will benefit by this study and, in turn, contribute their experiences with data dictionaries to the common body of knowledge.

ACKNOWLEDGEMENT

The authors wish to thank Steve Beasley and Ken Knepp for the considerable time and effort they have contributed to this project. The typing was done by Maxine Loeber whose accuracy is greatly appreciated. Finally, support for this project by C. M. Funk & Co. and Anderson College is gratefully acknowledged.

APPENDIX A

INITIAL MAPPING OF THE COMPREHENSIVE DATA DICTIONARY INTO ITSELF USING RELATE/3000

FILE: DATATYP

ITEMS: DATATYP	BITS	MIN	MAX	CHECK FIX INEX EXIN
INTEGER	16	-32768	32767	ASCII BINARY
REAL	32	$-1.15792*10^{76}$	$1.15792*10^{76}$	'INEXT 'EXTIN
LONG	64	$-1.15792*10^{76}$	$1.15792*10^{76}$	'INEXT 'EXTIN
BYTE	8	0	255	
LOGICAL	16	0	65535	ASCII BINARY
DOUBLE	32	-2147483648	2147483648	DASCII DBINARY

FILE: ITEM

ITEMS: ITEM	DATATYP	OCCUR	OCCUR	OCCUR	DEFAULT	UNIT
DATATYP	BYTE	8	1	1		
BITS	INTEGER	1	1	1		
MIN	LONG	1	1	1	0	
MAX	LONG	1	1	1	0	
CHECK	BYTE	8	1	1		
FIX	BYTE	8	1	1		
INEX	BYTE	8	1	1		
EXIN	BYTE	8	1	1		
ITEM	BYTE	12	1	1		
KEY	BYTE	16	1	1		
POSITION	INTEGER	1	1	1	1	
FILE	BYTE	26	1	1		
TYPE	BYTE	8	1	1		
UNIQUE	BYTE	1	1	1	T	
OWN_FILE	BYTE	26	1	1		
MBR_FILE	BYTE	26	1	1		
PROGRAM	BYTE	26	1	1		
ACCTYPE	INTEGER	1	1	1		
APPLICA	BYTE	8	1	1		
TYPEFUN	BYTE	8	1	1		
OCCUR1	INTEGER	1	1	1	1	
OCCUR2	INTEGER	1	1	1	1	
OCCUR3	INTEGER	1	1	1	1	
DEFAULT	BYTE	8	1	1		
UNIT	BYTE	8	1	1		
RECORD	BYTE	16	1	1		
FILETYP	BYTE	8	1	1		
FUNCTION	BYTE	8	1	1		
FORMAT	BYTE	20	1	1		
GROUP	BYTE	16	1	1		
DEV_CLASS	BYTE	8	1	1		
USER	BYTE	8	1	1		
CONTROL	BYTE	8	1	1		
PREDICATE	BYTE	28	1	1		
FTHRGRP	BYTE	16	1	1		
SONGRP	BYTE	16	1	1		
PASSWORD	BYTE	8	1	1		

FILE: RECORD

ITEMS: RECORD

RC-DATATYP	RC-KEYITEM
RC-TYPEFUN	RC-PGMACC
RC-ITEM	RC-ACCFUN
RC-RECORD	RC-FILETYP
RC-RECFMT	RC-ITEMACC
RC-FILE	RC-PROGRAM
RC-GROUP	RC-APPLICA
RC-GRPFMT	RC-ACCTYPE
RC-SELRULE	RC-FILLACC
RC-RLTNSHP	RC-USERAPP
RC-KEY	RC-USERCLS

FILE: RECFMT

ITEMS: RECORD	ITEM	POS	FORMAT
RC-DATATYP	DATATYP	1	
RC-DATATYP	BITS	2	
RC-DATATYP	MIN	3	
RC-DATATYP	MAX	4	
RC-DATATYP	CHECK	5	
RC-DATATYP	FIX	6	
RC-DATATYP	INEX	7	
RC-DATATYP	EXIN	8	
RC-TYPEFUN	TYPEFUN	1	
RC-TYPEFUN	DATATYP	2	
RC-ITEM	ITEM	1	
RC-ITEM	DATATYP	2	
RC-ITEM	OCCUR1	3	
RC-ITEM	OCCUR2	4	
RC-ITEM	OCCUR3	5	
RC-ITEM	DEFAULT	6	
RC-ITEM	UNIT	7	
RC-RECORD	RECORD	1	
RC-RECFMT	RECORD	1	
RC-RECFMT	ITEM	2	
RC-RECFMT	POSITION	3	
RC-RECFMT	FORMAT	4	
RC-FILE	FILE	1	
RC-FILE	FILETYP	2	
RC-FILE	RECORD	3	
RC-GROUP	GROUP	1	
RC-GROUP	FILE	2	
RC-GRPFMT	RECORD	1	
RC-GRPFMT	GROUP	2	
RC-GRPFMT	CONTROL	3	
RC-GRPFMT	POSITION	4	
RC-SELRULE	PREDICATE	1	
RC-SELRULE	FUNCTION	2	
RC-SELRULE	FTHRGRP	3	
RC-SELRULE	SONGRP	4	
RC-RLTNSHP	OWN_FILE	1	
RC-RLTNSHP	MBR_FILE	2	
RC-RLTNSHP	ITEM	3	
RC-KEY	KEY	1	
RC-KEY	FILE	2	
RC-KEY	TYPE	3	
RC-KEY	UNIQUE	4	
RC-KEYITEM	ITEM	1	
RC-KEYITEM	KEY	2	
RC-KEYITEM	POSITION	3	
RC-PGMACC	FILE	1	
RC-PGMACC	PROGRAM	2	
RC-PGMACC	ACCTYPE	3	
RC-ACCFUN	FILETYP	1	
RC-ACCFUN	ACCTYPE	2	

FILE: RECFMT

ITEMS: RECORD	ITEM	POS	FORMAT
RC-ACCFUN	FUNCTION	3	
RC-FILETYP	FILETYP	1	
RC-FILETYP	DEV_CLASS	2	
RC-ITEMACC	RECORD	1	
RC-ITEMACC	ITEM	2	
RC-ITEMACC	USER	3	
RC-ITEMACC	ACCTYPE	4	
RC-PROGRAM	PROGRAM	1	
RC-PROGRAM	APPLICA	2	
RC-APPLICA	APPLICA	1	
RC-ACCTYPE	ACCTYPE	1	
RC-FILEACC	FILE	1	
RC-FILEACC	USER	2	
RC-FILEACC	ACCTYPE	3	
RC-USERAPP	USER	1	
RC-USERAPP	APPLICA	2	
RC-USERCLS	USER	1	
RC-USERCLS	PASSWORD	2	

FILE: FILE

ITEMS: FILE	FILETYP	RECORD
DATATYP	RELATE	RC-DATATYP
TYPEFUN	RELATE	RC-TYPEFUN
ITEM	RELATE	RC-ITEM
RECFMT	RELATE	RC-RECFMT
RECORD	RELATE	RC-RECORD
FILE	RELATE	RC-FILE
GROUP	RELATE	RC-GROUP
GRPFMT	RELATE	RC-GRPFMT
SELRULE	RELATE	RC-SELRULE
RLTNSHP	RELATE	RC-RLTNSHP
KEY	RELATE	RC-KEY
KEYITEM	RELATE	RC-KEYITEM
PGMACC	RELATE	RC-PGMACC
ACCFUN	RELATE	RC-ACCFUN
FILETYP	RELATE	RC-FILETYP
ITEMACC	RELATE	RC-ITEMACC
PROGRAM	RELATE	RC-PROGRAM
APPLICA	RELATE	RC-APPLICA
ACCTYPE	RELATE	RC-ACCTYPE
FILEACC	RELATE	RC-FILEACC
USERAPP	RELATE	RC-USERAPP
USERCLS	RELATE	RC-USERCLS

FILE: RLTNSHP

ITEMS: OWN_FILE	MBR_FILE	ITEM
DATATYP	TYPEFUN	DATATYP
DATATYP	ITEM	DATATYP
ITEM	RECFMT	ITEM
ITEM	KEYITEM	ITEM
RECFMT	ITEMACC	ITEM
RECORD	RECFMT	RECORD
RECORD	GRPFMT	RECORD
RECORD	FILE	RECORD
FILE	GROUP	FILE
FILE	PGMACC	FILE
FILE	RLTNSHP	OWN_FILE
FILE	RLTNSHP	MBR_FILE
FILE	KEY	FILE
KEY	KEYITEM	KEY
GROUP	GRPFMT	GROUP
GROUP	SELRULE	FTHRGRP
FILE	FILEACC	FILE
APPLICA	PROGRAM	APPLICA
PROGRAM	PGMACC	PROGRAM
APPLICA	USERAPP	APPLICA
USERCLS	USERAPP	USER
USERCLS	FILEACC	USER
USERCLS	ITEMACC	USER
ACCTYPE	FILEACC	ACCTYPE
ACCTYPE	ITEMACC	ACCTYPE
ACCTYPE	ACCFUN	ACCTYPE
ACCTYPE	PGMACC	ACCTYPE
FILETYP	ACCFUN	FILETYP
FILETYP	FILE	FILETYP

FILE: FILETYP

ITEMS: FILETYP	DEV_CLAS
SEQUEN	DISC
DIR-ACC	DISC
KSAM	DISC
IMAGE-MA	DISC
IMAGE-DE	DISC
RELATE	DISC
PROGRAM	DISC
JCL	DISC

FILE: ACCTYPE

ITEMS: ACCTY		
	128	384
	64	320
	160	416
	224	480
	240	496
	129	504
	65	508
	161	2
	225	258
	241	

FILE: ACCFUN

ITEMS: FILETYP	ACCTY FUNCTION
SEQUEN	128
SEQUEN	64
SEQUEN	160
SEQUEN	224
SEQUEN	240
SEQUEN	129
SEQUEN	65
SEQUEN	161
SEQUEN	225
SEQUEN	241
SEQUEN	384
SEQUEN	320
SEQUEN	416
SEQUEN	480
SEQUEN	496
SEQUEN	504
SEQUEN	508
DIR-ACC	128
DIR-ACC	64
DIR-ACC	160
DIR-ACC	224
DIR-ACC	240
DIR-ACC	129
DIR-ACC	65
DIR-ACC	161
DIR-ACC	225
DIR-ACC	241
DIR-ACC	384
DIR-ACC	320
DIR-ACC	416
DIR-ACC	480
DIR-ACC	496
DIR-ACC	504
DIR-ACC	508
KSAM	128
KSAM	64
KSAM	160
KSAM	224
KSAM	240
KSAM	129
KSAM	65
KSAM	161
KSAM	225
KSAM	241
KSAM	384
KSAM	320
KSAM	416
KSAM	480
KSAM	496
KSAM	504

FILE: ACCFUN

ITEMS: FILETYP	ACCTY FUNCTION
KSAM	508
IMAGE-MA	128
IMAGE-MA	64
IMAGE-MA	160
IMAGE-MA	224
IMAGE-MA	240
IMAGE-MA	129
IMAGE-MA	65
IMAGE-MA	161
IMAGE-MA	225
IMAGE-MA	241
IMAGE-MA	384
IMAGE-MA	320
IMAGE-MA	416
IMAGE-MA	480
IMAGE-MA	496
IMAGE-MA	504
IMAGE-MA	508
IMAGE-DE	128
IMAGE-DE	64
IMAGE-DE	160
IMAGE-DE	224
IMAGE-DE	240
IMAGE-DE	129
IMAGE-DE	65
IMAGE-DE	161
IMAGE-DE	225
IMAGE-DE	241
IMAGE-DE	384
IMAGE-DE	320
IMAGE-DE	416
IMAGE-DE	480
IMAGE-DE	496
IMAGE-DE	504
	508
RELATE	128
RELATE	64
RELATE	160
RELATE	224
RELATE	240
RELATE	129
RELATE	65
RELATE	161
RELATE	225
RELATE	241
RELATE	384
RELATE	320
RELATE	416
RELATE	480
RELATE	496

FILE: ACCFUN

ITEMS: FILETYP	ACCTY FUNCTION
RELATE	504
RELATE	508
PROGRAM	128
PROGRAM	64
PROGRAM	160
PROGRAM	224
PROGRAM	240
PROGRAM	129
PROGRAM	65
PROGRAM	161
PROGRAM	225
PROGRAM	241
PROGRAM	384
PROGRAM	320
PROGRAM	416
PROGRAM	480
PROGRAM	496
PROGRAM	504
PROGRAM	508
JCL	128
JCL	64
JCL	160
JCL	224
JCL	240
JCL	129
JCL	65
JCL	161
JCL	225
JCL	241
JCL	384
JCL	320
JCL	416
JCL	480
JCL	496
JCL	504
JCL	508

FILE: ITEMACC

ITEMS: ITEM	RECORD	ACCTY	USER
DATATYP	RC-DATATYP	508	MANAGER
BITS	RC-DATATYP	508	MANAGER
MIN	RC-DATATYP	508	MANAGER
MAX	RC-DATATYP	508	MANAGER
CHECK	RC-DATATYP	508	MANAGER
FIX	RC-DATATYP	508	MANAGER
INEX	RC-DATATYP	508	MANAGER
EXIN	RC-DATATYP	508	MANAGER
TYPEFUN	RC-TYPEFUN	508	MANAGER
DATATYP	RC-TYPEFUN	508	MANAGER
ITEM	RC-ITEM	508	MANAGER
DATATYP	RC-ITEM	508	MANAGER
OCCUR1	RC-ITEM	508	MANAGER
OCCUR2	RC-ITEM	508	MANAGER
OCCUR3	RC-ITEM	508	MANAGER
DEFAULT	RC-ITEM	508	MANAGER
UNIT	RC-ITEM	508	MANAGER
RECORD	RC-RECORD	508	MANAGER
RECORD	RC-RECFMT	508	MANAGER
ITEM	RC-RECFMT	508	MANAGER
POSITION	RC-RECFMT	508	MANAGER
FORMAT	RC-RECFMT	508	MANAGER
FILE	RC-FILE	508	MANAGER
FILETYP	RC-FILE	508	MANAGER
RECORD	RC-FILE	508	MANAGER
GROUP	RC-GROUP	508	MANAGER
FILE	RC-GROUP	508	MANAGER
RECORD	RC-GRPFMT	508	MANAGER
GROUP	RC-GRPFMT	508	MANAGER
CONTROL	RC-GRPFMT	508	MANAGER
POSITION	RC-GRPFMT	508	MANAGER
PREDICATE	RC-SELRULE	508	MANAGER
FUNCTION	RC-SELRULE	508	MANAGER
FTHRGRP	RC-SELRULE	508	MANAGER
SONGRP	RC-SELRULE	508	MANAGER
OWN_FILE	RLTNSHP	508	MANAGER
MBR_FILE	RLTNSHP	508	MANAGER
ITEM	RLTNSHP	508	MANAGER
KEY	RC-KEY	508	MANAGER
FILE	RC-KEY	508	MANAGER
TYPE	RC-KEY	508	MANAGER
UNIQUE	RC-KEY	508	MANAGER
ITEM	RC-KEYITEM	508	MANAGER
KEY	RC-KEYITEM	508	MANAGER
POSITION	RC-KEYITEM	508	MANAGER
FILE	RC-PGMACC	508	MANAGER
PROGRAM	RC-PGMACC	508	MANAGER
ACCTYPE	RC-PGMACC	508	MANAGER
FILETYP	RC-ACCFUN	508	MANAGER
ACCTYPE	RC-ACCFUN	508	MANAGER

FILE: ITEMACC

ITEMS: ITEM	RECORD	ACCTY	USER
FUNCTION	RC-ACCFUN	508	MANAGER
FILETYP	RC-FILETYP	508	MANAGER
DEV_CLASS	RC-FILETYP	508	MANAGER
ITEM	RC-ITEMACC	508	MANAGER
RECORD	RC-ITEMACC	508	MANAGER
ACCTYPE	RC-ITEMACC	508	MANAGER
USER	RC-ITEMACC	508	MANAGER
PROGRAM	RC-PROGRAM	508	MANAGER
APPLICA	RC-PROGRAM	508	MANAGER
APPLICA	RC-APPLICA	508	MANAGER
ACCTYPE	RC-ACCTYPE	508	MANAGER
FILE	RC-FILEACC	508	MANAGER
USER	RC-FILEACC	508	MANAGER
ACCTYPE	RC-FILEACC	508	MANAGER
USER	RC-USERAPP	508	MANAGER
APPLICA	RC-USERAPP	508	MANAGER
USER	RC-USERCLS	508	MANAGER
PASSWORD	RC-USERCLS	508	MANAGER

FILE: FILEACC

ITEMS: FILE	USER	ACCTY
DATATYP	MANAGER	508
TYPEFUN	MANAGER	508
ITEM	MANAGER	508
RECFMT	MANAGER	508
RECORD	MANAGER	508
FILE	MANAGER	508
GROUP	MANAGER	508
GRPFMT	MANAGER	508
SELRULE	MANAGER	508
RLTNSHP	MANAGER	508
KEY	MANAGER	508
KEYITEM	MANAGER	508
PGMACC	MANAGER	508
ACCFUN	MANAGER	508
FILETYP	MANAGER	508
ITEMACC	MANAGER	508
PROGRAM	MANAGER	508
APPLICA	MANAGER	508
ACCTYPE	MANAGER	508
FILEACC	MANAGER	508
USERAPP	MANAGER	508
USERCLS	MANAGER	508

The following files have trivial contents at this time, being either empty or having only one entry:

TYPEFUN	SELRULE	USERCLS	PGMACC
GROUP	PROGRAM	KEY	USERAPP
GRPFMT	APPLICA	KEYITEM	

Considerations for the Design of Quality Software

Jan Stambaugh
Hewlett-Packard Company
Business Computer Group

The design of software, like the design of hardware, is a science and not an art. Computer programming is a discipline and, as such, must be disciplined. There are specific procedures and considerations which can help to ensure reliable, high quality, software products.

What is meant by quality? Quality is the basic characteristic or property of something. A particular property could be physical, such as weight, size, and color, or chemical, such as composition. For example, the properties of a batch of chocolate fudge might be stated in terms of the hardness, the sweetness, the size and shape of the pieces, and so on. Any element that can be used to define the nature of a product can be called a quality characteristic.

How then can we define software quality? Software quality can be looked at in two ways: first, the quality of the design itself and, second, the quality of the conformance to the design. The first classification deals with the degree of excellence of the ideas which define the product design. The second classification relates to the degree of excellence to which the product conforms to the design specifications.

The goal in software development is to create software that performs reliably, meets user requirements, and does nothing that it is not supposed to do. However, there is currently no standard definition of what qualities should be considered in developing software. There is no standard means for measuring quality quantitatively. Because the production of software is finally making the transition from an art to an engineering discipline, quality is now being given more objective and less subjective consideration. While the current state-of-the-art in software design imposes specific limitations on our ability to automatically measure quality, software researchers and developers are beginning to find ways to evaluate the quality of software quantitatively.

How can quality be designed into software? There are three primary considerations. The first is the establishment and adherence to standards for the use of a (1) STRUCTURED SYSTEMS DEVELOPMENT METHODOLOGY.

At Hewlett-Packard, software products developed by the R&D labs follow a product lifecycle plan. Many of the elements of this plan are appropriate and relevant to the design of applications software by our users.

The software product lifecycle plan stresses the importance of scheduling by first defining all the tasks which need to be done. It suggests that software projects are frequently late because the amount of work required to complete the entire project is not made obvious from the start. A software project must be divided into as many identifiable operations as possible. The product lifecycle helps to simplify scheduling by identifying the milestones which are common to all software projects and by explaining how to determine other factors which may be unique to a particular project.

Documentation is necessary for the long term success of any software effort. The product lifecycle proposes a standard for documentation which defines what needs to be recorded as well as when it should be. It also addresses the archiving of these documents so that they can be easily referenced at a later date.

The project review cycle advocated by the lifecycle plan defines formalized sign-offs during the progression of a product's development from one phase to the next. The review cycle provides a check and balance function to ensure that what is being developed is in fact what will be needed.

The lifecycle plan describes two types of project classifications for every software project. These classifications provide guidelines to the project management team in establishing priorities. The first classification deals with the value of the project relative to all concurrent efforts in each of the functionally responsible areas. The second classification is a development category. Each project is tagged as an enhancement, a new product, or the conversion of an existing product. This second classification conveys the extent to which previous projects may reduce the development effort.

The software product lifecycle stresses an iterative design philosophy. The design of a product is continually refined throughout the product's pre-release cycle. Rather than finalizing the external specifications and then beginning the internal design, the software product lifecycle proposes that both be outlined early on and refined as the project progresses. This allows the project team to maintain a global perspective of both sides of the development effort.

Project review policies and procedures are defined explicitly in the product lifecycle plan. A lab review team, chosen by lab management, is responsible for the evaluation of one specific project from start to finish. The product team members are chosen from marketing, manufacturing, the R&D lab, and product assurance and the responsibilities of each member are defined in detail.

Each project utilizes the software development network, a generalized PERT-like scheduling tool designed to aid in project planning and control. It provides an overview of the milestones to be met and an indication of their relative timing.

Each software project goes through five phases: investigation, design, implementation, testing, and release. During the investigation phase, a proposal is developed which describes a possible area for a software contribution, suggests a software project team, and sets a date for reviewing the research findings. The investigation report describes the results of the research and identifies pertinent issues which have a bearing on the decision of whether to continue with the development of a product. A product datasheet is generated to provide a quick overview of the product's key points. During the investigation phase sign-off review, the fact that a complete investigation has been conducted by the product team is verified. A decision can now be made as to whether or not to proceed to the design phase.

The design phase of product development is where the product is defined in detail. Two major documents are produced during this phase: the design outline and the external specifications. The design outline is a first draft description of the internal structure necessary to implement the product. It establishes the basic system modules and identifies key shared data structures and tables. It also establishes a plan for the detailed design of the product. The external specifications describe the functions of a product and how to use it. They provide the basis for the internal design of the product. The external specifications include hardware requirements and restrictions, software requirements and restrictions, user documentation requirements, detailed functional specifications, individual function descriptions, the user interface, compatibility specifications, security specifications, installation instructions, performance predictions, reliability/recoverability specifications, special capabilities and features, and error messages, meanings, and actions. A resource and schedule summary is created at this stage, outlining the financial cost of the project at each major milestone and and updating previous estimates for project completion. The design phase sign-off review provides a formal review of the external characteristics of the proposed product to ensure that the product team is in agreement that what is being proposed is in fact the appropriate solution to the problem being addressed.

The implementation phase involves the creation of the internal design document. This document describes in detail the algorithms and the data structures to be used in implementing the product. It serves as the internal documentation for the product throughout the remainder of its lifecycle. During the implementation phase, test sites are selected. There are three types of test sites: Alpha, Beta, and foreign language. An Alpha site is internal to HP; a Beta site is typically external; a foreign site may be either, depending on the scope of the project. For a product to go to Alpha test, it must be functionally complete and have very few known bugs. For a product to go to Beta test, there must be no known bugs that would seriously impede the user, and the preliminary documentation must be complete. Just prior to the development phase sign-off review, the product team is responsible for making a presentation to the lab which serves as a brief introduction to the use of the product. Once again the resources and schedule summary is updated to outline the financial cost of the project at this stage and to establish updated estimates for project completion.

A product cannot be developed and then tested. Consequently, testing is not really a phase but an integral part of the development and release of a high quality software product. The goal of testing is to uncover errors and deficiencies at the earliest possible moment, thus eliminating the possibility of fatal surprises. The majority of the test effort is aimed at ensuring that the end product fulfills the original specifications and that the particular implementation of the product is well executed. Many different testing techniques and processes are used. A code inspection is a set of procedures and error-detection techniques for group code reading. The general procedure involves the distribution of a program listing and related design specifications to participants several days in advance of the inspection session. The programmer narrates, statement by statement, the logic of the program. Questions are raised to determine if errors exist. The program is analyzed with respect to a checklist of historically common programming errors. The errors identified are also analyzed and used to refine the error checklist to improve the effectiveness of its future use.

The structured walkthrough, like the inspection, is a set of procedures and error-detection techniques for group code reading. Rather than reading the program or using error checklists, the walkthrough participants "play computer." A person who has been designated as the tester comes to the meeting armed with a small set of paper test cases for the program or module. During the meeting, each test case is mentally executed.

Module testing, or unit testing, is a process of testing the individual subprograms, subroutines, or procedures in a program.

Incremental testing or integration is a method of combining the next module to be tested with the set of previously tested modules before it is tested. Incremental testing has two strategies: top-down testing and bottom-up testing. The top-down strategy starts with

the initial module in the program. The rule for the next module to be eligible as the next module to be tested is that at least one of the module's calling modules must have been previously tested. The bottom-up strategy starts with the terminal modules in the program, the modules which do not call other modules. Here the rule for a module to be eligible as the next module to be tested is that all of the modules it calls must have been tested previously. Top-down testing requires the generation of stub modules and bottom-up testing the generation of driver modules.

Function testing is the process of attempting to uncover discrepancies between the program or system and its external specifications. It is not intended to check out the interactions between functions, but rather the functions themselves.

System testing is a process used to compare the system or program to its original objectives. It is a set of tests to verify that all components work together harmoniously. System testing includes the following: facility testing, volume testing, stress testing, usability testing, security testing, performance testing, storage testing, configuration testing, compatability/conversion testing, installability testing, reliability testing, recovery testing, serviceability testing, documentation testing, and procedure testing.

Acceptance testing is the process of comparing the program to its initial requirements and the current needs of its end users. This testing is accomplished by the test sites.

Installation testing takes place as a means of finding installation errrors. The test cases check to ensure that a compatible set of options has been selected, that all parts of the system exist, that all files have been created and contain the necessary contents, and that the hardware configuration is appropriate.

The testplan is a crucial part of the testing process. It outlines the types of test which will be used by the project team and specifies any non-standard tests, as well as the frequency to be used for repetitive operations such as walkthroughs. It should define a set of tests which will be sufficient to guarantee the quality of the finished product upon release from the lab. The testplan is developed as an integral part of the design and implementation phases, and not as an afterthought.

The automated test specification documents individual test programs, data sets, and procedures. It describes the purpose of each test as well as providing some detail about the internal workings.

The release phase involves reliability certification, performance specification and tuning, and turning control of the product over to manufacturing. Up until this phase, performance has been an issue of prediction and calculation. Now it is an issue of measurement. What the end user will see must be quantified, and the internal quantities which might affect these tangible values must be identified. The manufacturing release sign-off review establishes the product's completion and readiness for sale and distribution, and signals its release from the R&D lab and its entry into the maintenance phase of the product lifecycle. Six months after the product has been released to the field, a post release review meeting is called for the purpose of reflecting on the acceptability of the product in the marketplace. All members of the product team are required to attend.

The second consideration for the design of quality software is the use of (2) QUALITY-ENHANCING TOOLS AND TECHNIQUES. Some of the tools for enhancing quality are: database management systems, data dictionaries, report generators, graphics products, software monitors and optimizers, flow analyzers, cross-reference generators, languages, preprocessors, debugging software, program and test data libraries, an interactive programming facility, file maintenance systems, and project management systems.

Some of the techniques which can be utilized to enhance quality include structured walkthroughs, structured testing, development support libraries, excessive training, follow-on consulting, project audits, and software quality checklists.

The third and perhaps the most important consideration for ensuring the design of quality software is the (3) INTELLIGENT MANAGEMENT OF PEOPLE. The best structured systems development methodology combined with the best tools and techniques will be of little value if there is a lack of appropriate management.

Demonstrate leadership by example. Show that you care. Clearly define and communicate strategy and goals, but don't define every tactical step. Encourage your employees to participate in planning tasks. Allow them to be a part of the decision-making process. Ensure that they have adequate professional training. Provide them with feedback and recognition. And develop a constructive spirit of teamwork and cooperation.

To summarize, the three factors which most greatly influence the design of quality software are: the use of a structured systems development methodology, the use of quality-enhancing tools and techniques, and the intelligent management of people.

BIBLIOGRAPHY

Cho, Chin-Kuei. An Introduction to Software Quality Control. John Wiley & Sons, Inc., 1980.

Crowley, John D. "The Application Development Process: What's Wrong With It?" 1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality. JDC Associates, 1979.

McCall, James A. "An Introduction to Software Quality Metrics." In Concepts of Software Quality, 1978.

Myers, Glenford J. Software Reliability Principles and Practices. John Wiley & Sons, Inc., 1976.

Myers, Glenford J. The Art of Software Testing. John Wiley & Sons, Inc., 1979.

Welburn, Tuler. Structured Cobol Fundamentals and Style. Mayfield Publishing Company, 1981.

Zachmann, William F. Keys to Enhancing System Development Productivity. Amacom, 1981.

Characteristics of Software Quality. TRW Systems and Energy, Inc., North-Holland Publishing Co., 1978.

"Program Design Techniques." In EDP Analyzer, March, 1979, Vol. 17, No. 3.

"The Production of Better Software." In EDP Analyzer, February 1979, Vol. 17, No. 2.

Software Product Lifecycle. Hewlett-Packard Company, March, 1981.

LOOK/3000
A New Real-Time System Performance Monitoring Tool

Kim D. Leeper
Wick Hill Associates Limited

INTRODUCTION

All programs resemble one another. This might seem to be a rash statement but let us examine the facts. Programs were designed to perform the same task over and over again; in order to do this, one must design the program to iterate through a set of data. This is true even on interactive programs. Each screen could be thought of as an input step in preparation for future major loop in the system. This paper will deal with how to identify these loops without having seen the application code.

If the reader directs his/her attention to Figure 1, s/he will see a generic application flowchart. The application in question has five loops labelled A, B, C, D and E. The number of times each loop is executed is also noted beside each loop. In order to transform this generic application flowchart into a program, the chart must be turned into a linear diagram. A linear diagram is required because a computer executes a single thread of instructions. A programmer's job is to be able to translate the two dimension flowchart into a linear series of instructions. This diagram may be seen in Figure 2.

If we were to examine the execution of this generic program over a period of time, we would find that the amount of time spent at a given location of memory would be proportional to the number of times we executed the corresponding program loop. This is graphically demonstrated in the time graph of Figure 3.

Some obscure law of computing, probably one of Murphy's Laws, tells us that the linear diagram as described above is going to be too long to fit inside the physical constraints of the computer we are programming for. This restriction presents us with an interesting problem — how to divide up the program so we can execute it on our machine. On the HP3000 this act of dividing the program up is called segmentation. Before delving into segmentation in depth, let us examine a mathematical/graphical explanation of the subject of locality.

What is Locality?

Locality is a measure of how well segmented your program is. I define it as the ratio of the number of internal PCALs to the number of external PCALs on a percentage basis. In equation form it would be the following:

$$\frac{\text{number of internal PCALs in a segment}}{\text{number of external PCALs in the same segment}}$$

The word PCAL stands for procedure call. It is the instruction generated by the language compiler when, in your application program, a CALL is made to a library routine in COBOL, a SECTION is PERFORMed or a named procedure is executed in SPL. See Figure 4 for visual assistance in understanding this concept.

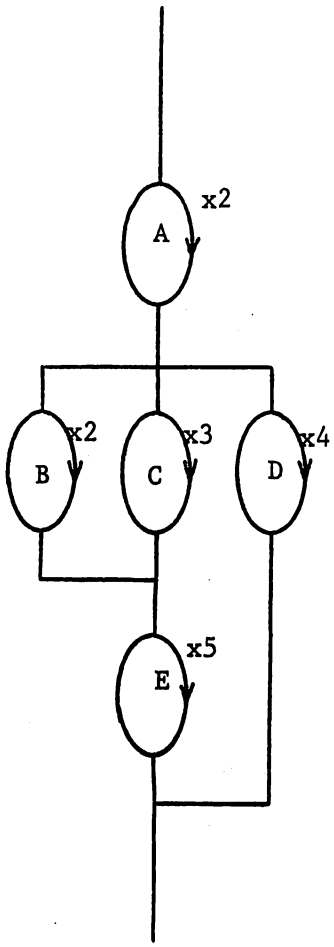
Why is Locality Important?

I am sure that you have all heard the following comment on segmentation: "Once in a segment stay there; Once out stay out." This comment is very appropriate to the HP3000 because of the significant difference in execution time between an internal PCAL and an external PCAL. Once you get into a segment, the program is advised to stay there because internal PCALs are twice as fast in execution compared with external PCALs. An internal PCAL/EXIT pair takes about 13 micro-seconds to execute on a Series II/III. An external PCAL/EXIT pair takes approximately 27 micro-seconds to execute on a Series II/III if the target code segment is in memory. If the code segment has to be read externally from the disk, then a disk access has to be added which brings the execution time up to 35-45 milli-seconds. This is obviously a significant time difference.

It is now possible to appreciate that ill-advised segmentation can have a significant adverse effect on the performance of the application program in question. You should aim to program for a maximum number of internal PCALs per segment. If you don't, you will waste time. To optimize your system from the time point of view, you must therefore segment your program appropriately.

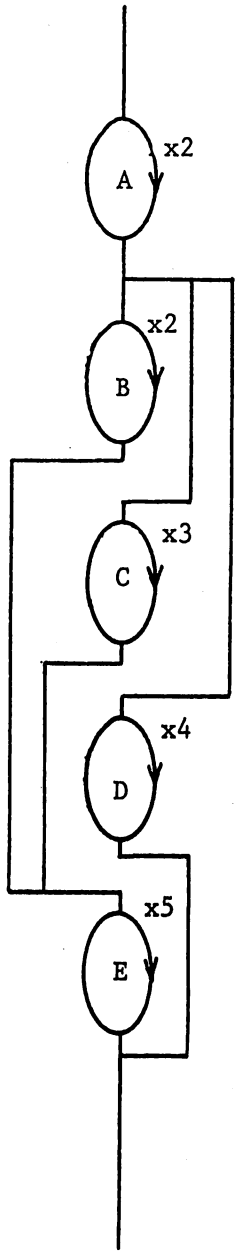
How Do You Identify If Your Program Is Appropriately Segmented?

One can use four different methods to identify if your program is properly segmented. The methods are as follows:



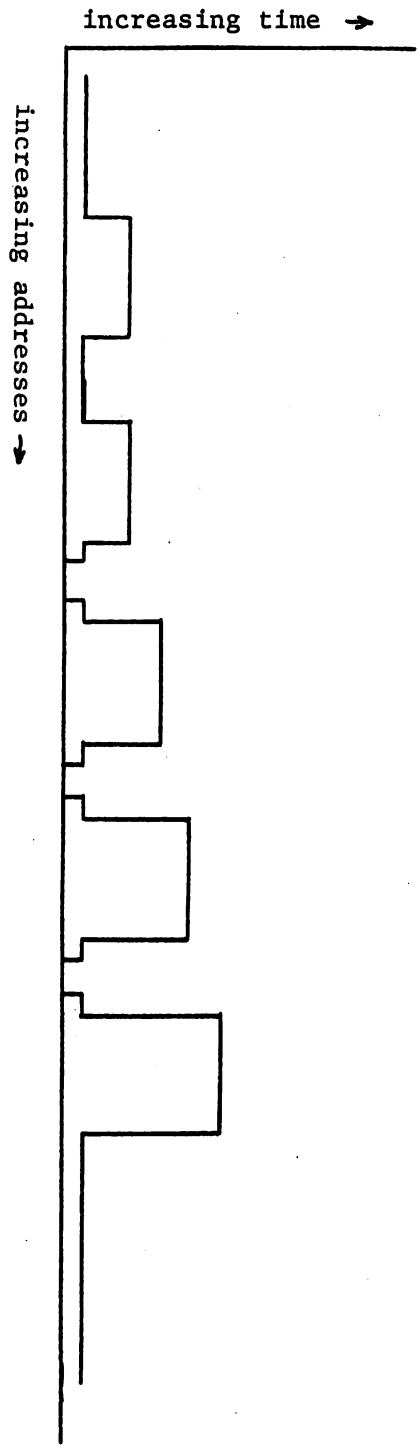
generic application flowchart

Figure 1



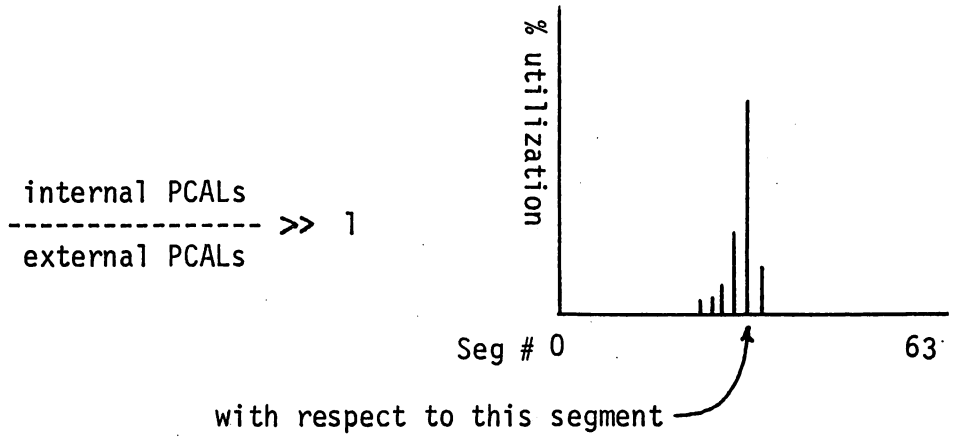
linear flow diagram

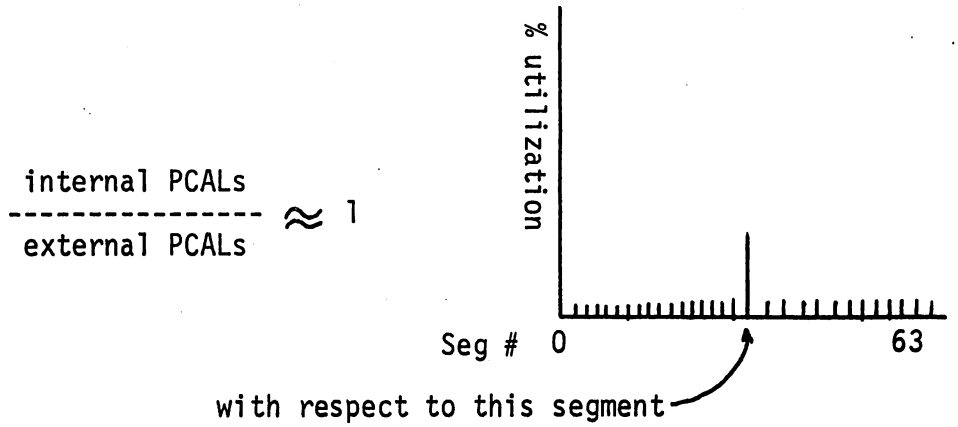
Figure 2

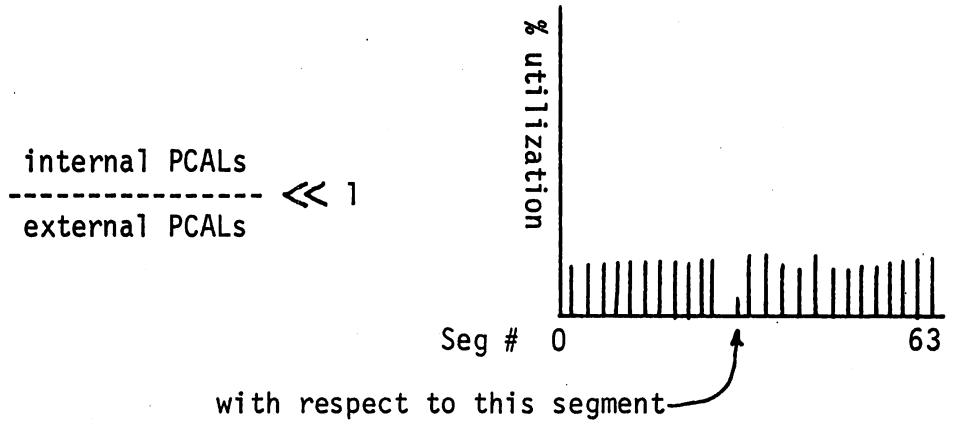


time graph

Figure 3







Locality Diagrams

Figure 4

1. use PROGSTAT out of the contributed library
2. use your eye
3. use programmer placed counters in the code
4. use LOOK/3000

PROGSTAT is an interesting program designed to provide a picture of how a program is segmented. It produces a list of external system references that the program under examination calls. It gives a separate count of external segment references that are satisfied within the program file itself. It also gives the segment lengths both actual and in a graphic form so one can balance the code lengths. PROGSTAT, unfortunately, does not provide a count of internal calls per segment so the designer may not calculate the locality profile as described earlier. PROGSTAT does not provide enough information to properly resegment one application program.

The oldest method available to the designer to segment his/her application code is to use the eye. This technique is dependent upon the experience of the individual using it. It is prone to error. The technique is time consuming. It is always biased. The designer might not realize that the data flow occuring in real life is not the way s/he imagines it to be. If the designer does not correctly segment the program for the flow of data emanating from the user, then the program might as well not be segmented. But how does the designer determine the character of the data flow? This brings us to the next method.

Placing counters in the code is a way of gathering information to help in the determination of appropriate segmentation. By judicious placement of the counters, we may determine an execution profile to assist us in proper segmentation. However, this technique is fraught with problems as well. The counters need to be initialized; they will need debugging; they will require stack space, and they will interfere with execution. The worst problem they introduce is that in a tight application program, placing counters in the code might require resegmentation just to get the program to run.

If we can use this method, we do gather some information as to how the data flow is causing the program to execute in a particular number of segments. The data from the counters could be used to draw some locality diagrams to assist in the resegmentation process.

The last method of determining if your program is segmented correctly is to use a system called LOOK/3000. This is a software tool provided by Wick Hill Associates Limited. LOOK produces the locality diagrams as shown on the next few pages. The biggest advantage LOOK provides is to allow the designer to watch the way the application is really being used with real data in real time, so when s/he resegments the application s/he has some assurance that the segmentation corresponds to the way the program is really being used. See Figures 5-9 regarding the displays LOOK produces.

Figure 5 is the display which allows the user to pick out the program/process that s/he wishes to examine in depth. This display is called the "SPECIFY PROGRAM SCREEN." The user identifies the process that s/he wants to examine by noting the PIN of that process. The PIN is known as the process identification number and is the number by which MPE manages your program. This PIN is entered and LOOK starts to acquire data regarding the process so identified. Every 10 seconds the display is updated. After the user enters the PIN of the process to be examined, the next screen is shown.

The next screen is called the "SEGMENT MAP SCREEN." This screen may be seen in Figure 6. It is the Locality Diagram of the program which was identified in the previous screen. This display helps the user to gain an understanding of the interaction between segments that make up the program under examination. This screen will be updated every 10 seconds. The user is now required to choose which segment s/he wishes to examine more closely. In this example segment number 5 was chosen.

The display shown next is Figure 7. It is called the "UNIQUE SEGMENT SCREEN." It is an overall map of CPU activity in this segment. In this example the user was only able to get two data points in this segment. Had the user waited longer, s/he would have acquired more data regarding segment number 5. The display shows that since the user chose to examine this segment, the CPU has executed two instructions in this segment. The first instruction is located between %1000 and %1377. The second instruction is located between %2400 and %2777. The user may choose any location to

	FILE NAME		USER NAME		Q	J/S#	STACK SIZE	PIN#	NUM SEGS	CPU TIME
1	ENTRY .PUB	.SYS	SIMON	.GOLUB	C	S389	4924	16	1	144
2	ORDRNTRY.SKORDS	.GOLUB	SIMON	.GOLUB	C	S388	17036	20	42	0
3	ORDRNTRY.SKORDS	.GOLUB	SIMON	.GOLUB	C	S364	17164	22	42	271
4	C.I.<LISTF		) SIMON	.GOLUB	C	S407	3036	24	0	0
5	SAR1350 .PUB	.GOLUB	SIMON	.GOLUB	D	J212	3612	27	2	4643
6	APG3 .PUB	.WHA	MANAGER	.WHA	C	S406	23684	30	4	413
7	C.I.<REPLY 30,7		) MANAGER	.SYS	C	S380	2140	32	0	0
8	ENTRY .PUB	.SYS	SIMON	.GOLUB	C	S403	4924	33	1	62

Figure 5

```
   PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB          TOTAL NUMBER OF SEGMENTS = 42

  SEGMENT NUMBER = %05                              LENGTH OF SEGMENT = %003130

  ADDRESS = %002400

  THIS PROGRAM WAS RUN ON A SERIES III

          100|              *
          95 |              *
          90 |              *
          85 |              *
          80 |              *
          75 |              *
          70 |              *
          65 |              *
  % TIME  60 |              *
  SPENT   55 |              *
  AT      50 |              *
  ADDRESS 45 |              *
          40 |              *
          35 |              *
          30 |              *
          25 |              *
          20 |              *
          15 |              *
          10 |              *
          5  |              *
  LOCATION IN+-----------------------------------------------------------------
  SEGMENT ->  0000000011111111222222223333333344444444555555556666666677777777
    002400 +  0123456701234567012345670123456701234567012345670123456701234567

   TOTAL NUMBER OF SAMPLES IS 1
```

Figure 6

```
   PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB          TOTAL NUMBER OF SEGMENTS = 42

  SEGMENT NUMBER = %05                              LENGTH OF SEGMENT = %003130

  ADDRESS = %002400

  THIS PROGRAM WAS RUN ON A SERIES III

          100|
          95 |
          90 |
          85 |
          80 |
          75 |
          70 |
          65 |
  % TIME  60 |
  SPENT   55 |
  AT      50 |   *  *
  ADDRESS 45 |   *  *
          40 |   *  *
          35 |   *  *
          30 |   *  *
          25 |   *  *
          20 |   *  *
          15 |   *  *
          10 |   *  *
          5  |   *  *
             +-----------------------------------------------------------------
  LOCATION IN                  1111111111111111222222222222222233333333333333333
  SEGMENT  -> 0011223344556677001122334455667700112233445566770011223344556677
   % 100     0404040404040404040404040404040404040404040404040404040404040404

  TOTAL NUMBER OF SAMPLES IS 2
```

Figure 7

```
PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB          TOTAL NUMBER OF SEGMENTS = 42
SEGMENT NUMBER = %05                            LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

          100|                *
           95|                *
           90|                *
           85|                *
           80|                *
           75|                *
           70|                *
           65|                *
% TIME     60|                *
SPENT      55|                *
AT         50|                *
ADDRESS    45|                *
           40|                *
           35|                *
           30|                *
           25|                *
           20|                *
           15|                *
           10|                *
            5|                *
             +----------------------------------------------------------
LOCATION IN       0011223344556677        0011223344556677    IN
SEGMENT -> 002000+0404040404040404 003000+0404040404040404    OCTAL
           0000000000000000        0000000000000000

TOTAL NUMBER OF SAMPLES IS 1
```

Figure 8

```
PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB          TOTAL NUMBER OF SEGMENTS = 42
SEGMENT NUMBER = %05                            LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

          100|
           95|
           90|
           85|
           80|
           75|
           70|
           65|
% TIME     60|
SPENT      55|
IN         50|
SEGMENT    45|
           40|                                  *
           35|                                  *
           30|                                  *
           25|                                  *
           20|      *                           *
           15|      *           *               *
           10|      * *         *               *
            5|      * *   *     *               **
             +----------------------------------------------------------
SEGMENT #             1         2   3     4     5     6     7
FROM PMAP-> 0123456701234567012345670123456701234567012345670123456701234567

TOTAL NUMBER OF SAMPLES IS 18
```

Figure 9

look at but in this example s/he chose to examine the octal location 2400. This value is input and the user is now shown a more detailed display located around the location %2400.

This more detailed screen is called the "KILO-WORD WINDOW ON SEGMENT." Figure 8 shows this display very clearly. This display allows the user to determine if s/he wishes to continue examining this location in greater detail or whether s/he wants to move to another. The next screen shows this location at an expanded scale.

The new display is called the "CENTI-WORD WINDOW ON SEGMENT" screen. It gives such a detailed view of what is happening in the segment that the user may read off the actual address of where LOOK caught your application program. This is shown graphically in Figure 9. The address of the data point is %2415. This is where LOOK found the application program. If one examined the program over a longer period of time, then one could obviously get a much better picture of where the program under examination is spending its time.

By judicious use of LOOK/3000, one may locate down to the instruction address, where the application program is spending its time. Once this fact has been discovered what can be done about it?

ACTION PLAN

Three things can be done in order to improve the locality characteristics of an application program. These are as follows:

1. recode parts of the application code more efficiently
2. duplicate code modes by making them internal PCALs with judicious use of INCLUDES or COPYLIBS
3. resegment your application code

Where do we direct our attention to begin with? This is where the proprietary software tool called LOOK/3000 is invaluable. LOOK displays clearly where the application program is spending a large percentage of its execution time. As other authors have noted, programs spend 90% of their time executing 10% of the code. The trick is to identify which portion of the code you are spending your time in.

Once the offending part of the code is identified you could recode that part of your application. You might find your code executing a particular DBGET in the application program. Closer examination reveals the DBGET is acting on a data with a sorted chain that you thought had been removed months ago. You modify the schema to remove the sorted chain and the program now has a different profile because the application is not waiting for IMAGE to read down the chain.

Alternatively one could find the application code constantly calling another segment in your application. Closer examination would possibly indicate that the routine in constant use is a small one. The decision could be made to put this routine in a COPYLIB or INCLUDE file that could be inserted at compile time by the language translation. This action would make the code segment slightly larger but would remove an external PCAL.

LOOK/3000 can also help in the process of resegmentation of an existing program with an outdated design, i.e., where the application usage has changed.

CONCLUSION

As this paper has shown, poor segmentation on the HP3000 is an important contributor to poor application performance. We have seen various techniques to identify where a program is spending its time. We have seen the transformation between locality diagrams and segmentation. Various techniques have been offered regarding the evaluation of segmentation of a given application program. The most straightforward way of determining where you are spending your execution time is to use LOOK/3000. This is a proprietary software tool available from Wick Hill Associates, Ltd.

REFERENCES

1. Author unknown, "Segmentation for Maximum Efficiency of System Tape Programs," Communicator Number 5.
2. Author unknown, "Segmentation in COBOL," Communicator Number 12.
3. Author unknown, "Software Optimization Through Segmentation," Proceedings of IUG, February 1975.
4. Robert Green, "Principals for Optimizing Performance of On-Line Programs," HPGSUG Vol. II, No. 2, 1978.
5. Jim Squires and Ed Splinter, "System Performance Measurement and Optimization," Proceedings of IUG, November 1978.
6. Gerry Wade, "Programming for Survival," Proceedings of IUG, November 1978.
7. Rodney V. Smith, "Application Design for the HP3000," Proceedings of SCRUG, September 1980.
8. Robert Green, "HP3000/Optimizing Batch Jobs," Proceedings of IUG, April 1981.
9. Author unknown, "Application Design Course," HP Part # 22808A, November 1980.
10. Author unknown, "Application Design and Optimization for the HP3000," SE reference document, June 1978.
11. Jon W. Henderson, "Design and Segmentation Techniques for Large SPL Programs," Proceedings of IUG, February 1980.

QHELP: An On-Line Help System

David J. Greer
Robelle Consulting Ltd.

SUMMARY

QHELP is a software tool that provides an interactive help facility for on-line programs. QHELP is designed to be easy to set up and maintain, efficient at run time, and convenient for the end users. QHELP eliminates duplication by allowing the user manual and the help text to be the same file. A series of keywords defines a "tree" which is used to allow the end user easy access to any part of the help text.

Contents

INTRODUCTION

Most application subsystems and software utilities would benefit from an on-line HELP facility for users. QHELP is such a facility, and it does not require a great deal of system or programmer resources. QHELP makes it easy to maintain help text in an external QEDIT file.

For utility tools (i.e., QEDIT, SUPRTOOL, etc.), the HELP file would primarily explain the commands available in the tool, but would also give examples, uses, sample results and news of recent changes. One reason we developed QHELP was to allow Robelle software products to have a greatly expanded "HELP" capability.

For application programs, a HELP file would document what ach module in an application is supposed to do, in language the end user can understand. The benefits of using HELP files are: (1) they can be part of the specifications during the design stage of a project; (2) they assist in user training, eliminating many questions; (3) they are more accessible than written documentation and are much easier to keep up-to-date.

Ad hoc HELP systems tend to skimp on information for the user and are often out of date because they are not easy to maintain. A standard HELP system should be flexible, easy to program, low in overhead and hierarchical (many key levels), with many indices for quick retrieval by the user.

QHELP provides an easy means for the COBOL programmer (or the SPL or FORTRAN programmer) to code the HELP capability into an application program. The structure of the HELP text gives the user access to the information which is immediately necessary to the task at hand. In addition, QHELP is designed to consume a minimum of system resources when it isn't being used.

Why Not Use the MPE HELP Facility?

The MPE HELP facility has several problems that make it awkward to use. One problem is that the HELP text can only be organized on two levels. Many applications will be easier for the user to understand if the help material can be decomposed into more than two levels of detail. "Subdivision" should make it easier to maintain the documentation (increasing the probability that maintenance will actually be done).

The MPE HELP system is not designed to be used from a COBOL program. Without writing some SPL interfaces, it is not possible for the COBOL programmer to access the MPE HELP facility. In addition, the MPE HELP files must be updated with user documentation every time an update is done to MPE.

With the MPE HELP facility, HELP text cannot be broken down into different files. Ideally, every programmer should maintain the HELP documentation for each module/program that he writes. This is done most easily by having one HELP file per module/program; but it should be easy to connect all of the HELP files and make them look like an integrated whole to the user.

Is QHELP Any Better?

QHELP solves all of the problems mentioned above. Within QHELP, entries can be organized into as many as ten levels of keyword indices. QHELP keeps the help files open and saves indices at every level of the HELP file so that searches are fast. QHELP uses QEDIT work files, which consume less disc space and are more efficient than regular KEEP files. QHELP provides modular files (similar to $INCLUDE) so that a system of HELP files can be easily integrated into one HELP package. Finally, the programmatic interface to QHELP is flexible. The *FIND* and *UNFIND* commands, for example, allow an application subprogram to position the HELP file pointers to the HELP text for that subprogram, without knowing anything about

"higher" levels in the HELP file (i.e., QHELP provides relative indexing).

QHELP is distributed to all users of Robelle software products as part of the QLIB contributed library (no extra charge). Users are authorized to merge QHELP into their own software, without restriction. QHELP works only with QEDIT files. There are four reasons for this: (1) QEDIT files provide data compression (reducing the cost of the help facility); (2) QEDIT files can be easily and efficiently modified (if you have QEDIT); (3) QEDIT files provide fast random access (as required for the recursive tree structure); and 4) we may sell more copies of QEDIT by tying QHELP to it.

STRUCTURE OF A QHELP FILE

The basic structure of a QHELP file consists of a key (e.g., QEDIT), some text about the key, and, optionally, lower level keys (e.g., NEWS, Add, Change . . .). The structure is recursive: lower level keys can also have text and more keys. A new key is specified by the \BEGINKEY command. The end of a key is indicated by the \ENDKEY command. Everything between the \BEGINKEY command and the \ENDKEY command is considered to be part of the key.

The following is an extract from an early HELP file for QEDIT:

```
\BEGINKEY QEDIT

                QEDIT Capsule Summary

/S LANG=COB         Select COBOL as the current 'language'.
/TEXT SRC2=SRC1     Make a new QEDIT file SRC2 and copy SRC1 into it.
/OPEN SRC2          Select SRC2 as the current workfile for editing.
/LIST 3/13          Display lines from SRC2 (current workfile).
/LIST SRC1 5/10     List lines 5 thru 10 of SRC1 (not current file).
/LIST \BAL-DUE\     List all lines of the workfile with BAL-DUE in them.
/MOD [ ];D 63.5     Modify FIRST and LAST ([ ]) lines, delete line 63.5.
/ADD 5              Add new lines to workfile at or after line 5.
/ADD 50 = 70/78     Make a copy of lines 70 thru 78 after line 50.
/ADD 40 < 20/30     Move lines 20 thru 30 to after line 40.
/ADD 20 = TXT4      Copy all of the file TXT4 after line 20.
/C "XY"ABC"@        Change each string "XX" to "ABC" in ALL lines.
/:COBOL *           Compile the current workfile, listing on terminal.
/:PREP              Preps $OLDPASS into $NEWPASS; see /S OPT MAX.
/:RUN               Runs $OLDPASS (result of :PREP); see /S OPT LIB.
/:STREAM *          Stream the /OPEN workfile as a batch job.
Special keys:       Control-Y (stop commands), Control-X (cancel input),
                    Control-S (suspend listing until Control-Q).
/HELP INTRO         Print more HELP (try /H NEWS, /H ADD, etc.).
/EXIT               Leave QEDIT and return to MPE for :BYE.

\BEGINKEY INTRO
              QEDIT:  Introducing Terms

  Term:                Explanation:
Workfile?              File built via NEW/TEXT that QEDIT edits (code=111)
                       QEDITSCR is the default, temporary file if none spec.
Current Workfile?      The QEDIT file that is currently /OPEN for editing.
External File?         Not the current workfile; QEDIT can /LIST any file.
Language?              Each file has 'lang' attached to it (COB,RPG,FTN,SPL,
                       JOB,TEXT for >80 columns,COBX to use comment field).
Command?               Defined by first letter (A=add), lower-case is okay,
                       and semicolon(;) sets off multiple commands per line.
        ADD       DELETE   GALLEY   LIST     OPEN     REPLACE  USE
        BEFORE    EXIT     HELP     MODIFY   PROC     SET      VERIFY
        CHANGE    FIND     KEEP     NEW      Q        TEXT     ZAVE
Command Option?     Second char modifies command action (LQ, AJ, KJ)
        T:  template option (prints column headings)
        Q:   quiet option (without linenumbers or without printing)
```

```
        J:   justified (/ADD), jumping (/L), window only (/K)
MPE Command?        Needs a colon(:), /:LISTF, /:STREAM, /:RUN, /:PREP.
Rangelist?          List of ranges (5/7,9) or string match ("bb" [range]).
Range?              Lines: 5, 5/7, @=ALL, [=FIRST, *=curr, 5/=5/LAST.
String?             Char in quotes("bb",-bb-,\bb\) to find or change.
Window?             Where/How to match strings: /S W=(1/50,SMART).
User Manual?        Enter /GALLEY QMANUAL.DOC.ROBELLE LP to print manual.

\ENDKEY INTRO
\BEGINKEY NEWS

     ...  NEWS about the latest version of QEDIT.

\ENDKEY NEWS

\BEGINKEY A

     ...  The ADD command is described here.

\ENDKEY A

     .
     .
     .

     The rest of the QEDIT commands are entered here.

\BEGINKEY S

     ...  A general description of the SET command goes here.

\BEGINKEY GENERAL

     ...  This key belongs to the SET command.  It describes all
          parts of the set command except for /SET OPTION.

\ENDKEY GENERAL

\BEGINKEY OPTION

     ...  Each of the /SET OPTION parameters is described here.

\ENDKEY OPTION

\ENDKEY S

\BEGINKEY Z

     ...  The ZAVE command is the last one in the HELP file.

\ENDKEY Z

\ENDKEY QEDIT
```

There are several important points to note: 1) The \BEGINKEY command and the \ENDKEY command may be spelled in any combination of upper- and lower-case letters. Within QHELP, all key names are upshifted. 2) The key for this HELP file is QEDIT (the first line in the file must contain a \BEGINKEY command). The keys that belong within QEDIT are INTRO, NEWS, A, . . . , Z. 3) Each \BEGINKEY command has a matching \ENDKEY command. The key name on the \BEGINKEY command and \ENDKEY commands must be identical.

Any key may have sub-keys associated with it. For example, the SET command above contains "GENERAL" and "OPTION" as keys. These two keywords are then associated with the SET command rather than with QEDIT.

Using QHELP with QGALLEY

Any lines in the HELP file that start with a backslash, but do not contain one of the HELP commands (BEGINKEY, ENDKEY, BEGININDEX, ENDINDEX), are retained, but ignored, by the QHELP system. This allows you to embed QGALLEY (a modified version of GALLEY) commands in the HELP file to specify formatting. Thus, the HELP file can be used as part of the user manual.

Example:

```
\BEGINKEY QEDIT
\FORMAT
    (text)
\IMAGE
    (list of commands)
\ENDKEY QEDIT
```

Any text between one \ENDKEY and the next \BEGINKEY will be ignored by QHELP, but will be picked up by QGALLEY and PROSE. A useful technique is to "include" the help file into the file for the user manual, thus separating the title page and the table of contents from the actual text. The OUTQ command of QGALLEY should be used to "cleanup" any help files that you have modified (in order to adjust lines so that they again fit neatly within the margins). Then, the file created by OUTQ can be processed through the QHELP compiler.

INTERACTING WITH QHELP

Assuming that you have RUN QHELP.QLIB and asked for the help file named QEDIT.HELP.ROBELLE, what happens next? QHELP prints the initial block of text from the help file (the text between \BEGINKEY QEDIT and \BEGININDEX INTRO, the first nested key word):

```
:HELLO user.account
:RUN QHELP.QLIB.ROBELLE

QHELP/QLIB/ROBELLE Consulting Ltd.(C) 1981
(Version 0.2)

Enter HELP Filename? QEDIT.HELP.ROBELLE

                QEDIT Capsule Summary

/S LANG=COB         Select COBOL as the current 'language'.
/TEXT SRC2=SRC1     Make a new QEDIT file SRC2 and copy SRC1 into it.
      .
      .
      .
/HELP INTRO         Print more HELP (try /H NEWS, /H ADD, etc.).
/EXIT               Leave QEDIT and return to MPE for :BYE.

LO: Keywords Under: QEDIT

INTRO,NEWS,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

```
>NEWS
```

QHELP prompts with a ">" character for the user to enter a keyword or a special command. If the user enters a valid keyword, all of the information for that keyword is displayed, along with a list of any sub-keys. In the example above, the user may type NEWS to get information on new features of QEDIT.

The user need not type the entire key word, since QHELP normally matches on any leading sub-string of the key (this feature can be disabled with a *SET* command). In this example, the user could have typed N, NE, or NEW to get NEWS (notice one disadvantage of this feature: the user cannot get to the N keyword for the \N command; some care must be given to the naming and ordering of key words).

Whenever QHELP is printing text, the user may strike Control-Y to interrupt the printout. QHELP will usually prompt for another key word.

The user leaves QHELP by typing 'exit', or by typing a circumflex (^), instead of a key name. Because the user may be several levels deep into the HELP file structure, he may have to type 'exit' several times (once for each level). Alternatively, several circumflexes (^) may be typed in a row; QHELP will return one level for each circumflex (e.g., entering >^^ exits two levels).

There are several special commands that may be entered when the user interacts with QHELP. One of these is the circumflex (^), which is used to exit the interact mode of QHELP. The question mark (?) prints information on the QHELP special commands. Another special command is the dollar sign ($), which is exactly the same as the QHELP *SET* command (e.g., $ LP ON equals SET LP ON).

Sometimes it is desirable to print all of the information under a specific level. The "at" sign (@) is a special

QHELP command (and key word) to do this.

The following example illustrates the use of these special commands. Starting at the node for QEDIT, output is directed to the line printer. Next, all of the information under QEDIT is printed on the line printer; then output is directed back to the terminal.

```
LO: Keywords Under: QEDIT

INTRO,NEWS,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
>$ LP ON

LO: Keywords Under: QEDIT

INTRO,NEWS,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
>@

LO: Keywords Under: QEDIT

INTRO,NEWS,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
>$ LP OFF
```

Suppose you want to see the help text associated with the key word GENERAL, nested under the key word S (for \SET command). One way to get this would be to enter the S keyword, wait for the S text and sub-keys to print, then enter the GENERAL key word, read the desired material and, finally, enter an "exit". Or, at the entry level to QHELP (i.e., level 0), you could enter both the key word and the sub-key, separated by commas; QHELP will follow the indicated path through the HELP file tree and print only the lowest level (then exit back to the original level).

```
>S,GENERAL
```

Suppose you want to see everything on the SET command. You can use the multiple key word option plus the "@" option to request all information under the S key word (Control-Y will bring you back to the original level immediately):

```
>S,@
```

USING QHELP FROM COBOL

QHELP is designed to make "help" information available to the end user with the minimum amount of involvement by the applications programmer. The COBOL programmer invokes QHELP by using a series of commands that are passed to the QHELP procedure through a standard communication area. The layout of the communication area is:

```
01  QHELP-AREA.
    05  QHELP-COMMAND              PIC X(80).
    05  QHELP-RESULT               PIC S9(4) COMP.
        88  QHELP-OK               VALUE ZEROS.
        88  QHELP-MISSING-KEY      VALUE 6.
    05  QHELP-BUFFER-LENGTH        PIC S9(4) COMP VALUE 200.
    05  FILLER                     PIC X(200).
```

To make maintenance easier, this area is normally declared in the COPYLIB, and copied into the source program with the COPY statement. All QHELP commands are moved into the QHELP-COMMAND buffer before calling QHELP, and QHELP returns the status of each call in the QHELP-RESULT variable. Like IMAGE, QHELP must be opened and initialized before it can be used. The *OPEN* command is used to start up the QHELP system, and to tell QHELP the name of the QHELP file. The following COBOL fragment opens a HELP file called EXAMPLE.HELP and initializes the QHELP system.

```
MOVE "OPEN EXAMPLE.HELP"          TO QHELP-COMMAND.
CALL "QHELP" USING QHELP-AREA.
IF NOT QHELP-OK THEN
    DISPLAY "ERROR:  CANNOT OPEN HELP FILE"
    MOVE FALSE                    TO QHELP-INITIALIZED-FLAG
ELSE
    MOVE TRUE                     TO QHELP-INITIALIZED-FLAG.
```

The applications program must check the user commands to see if the user requests help. When the user asks for help, the applications program must make one more call to QHELP, using the *PRINT* command.

```
MOVE "PRINT"                          TO QHELP-COMMAND.
CALL "QHELP" USING QHELP-AREA.
IF NOT QHELP-OK THEN
    DISPLAY "ERROR:  FAILURE OF QHELP ", QHELP-RESULT
    PERFORM END-OF-PROGRAM.
```

On the terminal, the result of the *PRINT* command would be as described above under "Interacting With QHELP".

The following example program demonstrates the use of QHELP. The program prompts the user for a command, and executes a separate module for each command. One of the valid commands is "HELP", and when the user types "HELP", QHELP is called.

```
$CONTROL SOURCE,ERRORS=5,LIST
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  EXAMPLE.
 AUTHOR.      DAVID GREER, ROBELLE CONSULTING LTD.

 ENVIRONMENT DIVISION.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  TRUE                          PIC X VALUE "T".
 01  FALSE                         PIC X VALUE "F".

 01  QHELP-INITIALIZED-FLAG        PIC X.
     88  QHELP-INITIALIZED         VALUE "T".

 01  ACCEPT-BUFFER                 PIC X(80).
     88  ANSWER-SPACES             VALUE SPACES.

 01  ACCEPT-BUFFER-1 REDEFINES ACCEPT-BUFFER.
     05  ACC-ONE                   PIC X.
         88  ACC-COMMAND-EXIT      VALUE "E".
         88  ACC-COMMAND-ADD       VALUE "A".
         88  ACC-COMMAND-CHANGE    VALUE "C".
         88  ACC-COMMAND-DELETE    VALUE "D".
         88  ACC-COMMAND-HELP      VALUE "H".
     05  FILLER                    PIC X(79).


 01  QHELP-AREA.
     05  QHELP-COMMAND             PIC X(80).
     05  QHELP-RESULT              PIC S9(4) COMP.
         88  QHELP-OK              VALUE ZEROS.
         88  QHELP-MISSING-KEY     VALUE 6.
     05  QHELP-BUFFER-LENGTH       PIC S9(4) COMP VALUE 200.
     05  FILLER                    PIC X(200).

 PROCEDURE DIVISION.
 00-MAIN                            SECTION.

     PERFORM 05-INITIALIZE
        THRU 05-INITIALIZE-EXIT.

     IF QHELP-INITIALIZED THEN
         MOVE SPACES                TO ACCEPT-BUFFER
         PERFORM 10-MAIN-PROCESSING
            THRU 10-MAIN-PROCESSING-EXIT
           UNTIL ACC-COMMAND-EXIT.

     PERFORM 95-FINISH-UP
        THRU 95-FINISH-UP-EXIT.
```

```
 00-MAIN-EXIT.  GOBACK.

$PAGE "[05]  INITIALIZE"
 05-INITIALIZE                      SECTION.

     MOVE "OPEN EXAMPLE.HELP"    TO QHELP-COMMAND.
     CALL "QHELP" USING QHELP-AREA.
     IF NOT QHELP-OK THEN
         DISPLAY "ERROR:  CANNOT OPEN HELP FILE"
         MOVE FALSE              TO QHELP-INITIALIZED-FLAG
     ELSE
         MOVE TRUE               TO QHELP-INITIALIZED-FLAG.

 05-INITIALIZE-EXIT.  EXIT.

$PAGE "[10]  MAIN PROCESSING"
 10-MAIN-PROCESSING             SECTION.

     MOVE SPACES                TO ACCEPT-BUFFER.

     PERFORM 10-10-GET-COMMAND
       UNTIL NOT ANSWER-SPACES.

     IF ACC-COMMAND-HELP THEN
         PERFORM 10-20-CALL-QHELP
     ELSE
     IF ACC-COMMAND-ADD THEN
         PERFORM 20-PROCESS-ADD-COMMAND
            THRU 20-PROCESS-ADD-COMMAND-EXIT
     ELSE
     IF ACC-COMMAND-CHANGE THEN
         PERFORM 30-PROCESS-CHANGE-COMMAND
            THRU 30-PROCESS-CHANGE-COMMAND-EXIT
     ELSE
     IF ACC-COMMAND-DELETE THEN
         PERFORM 40-PROCESS-DELETE-COMMAND
            THRU 40-PROCESS-DELETE-COMMAND-EXIT
     ELSE
      IF NOT ACC-COMMAND-EXIT THEN
         DISPLAY "ERROR:  UNKNOW COMMAND NAME".

     GO TO 10-MAIN-PROCESSING-EXIT.

 10-10-GET-COMMAND.
     DISPLAY "ENTER COMMAND NAME".
     ACCEPT ACCEPT-BUFFER.

 10-20-CALL-QHELP.
     MOVE "PRINT"               TO QHELP-COMMAND.
     CALL "QHELP" USING QHELP-AREA.
     IF NOT QHELP-OK THEN
         PERFORM 99-FATAL-ERROR
            THRU 99-FATAL-ERROR-EXIT.

 10-MAIN-PROCESSING-EXIT.  EXIT.

$PAGE "[20]  PROCESS ADD COMMAND"
 20-PROCESS-ADD-COMMAND         SECTION.

     DISPLAY "PROCESSING FOR THE ADD COMMAND".

 20-PROCESS-ADD-COMMAND-EXIT.  EXIT.

$PAGE "[30]  PROCESS CHANGE COMMAND"
 30-PROCESS-CHANGE-COMMAND      SECTION.
```

```
        DISPLAY "PROCESSING FOR THE CHANGE COMMAND".

    30-PROCESS-CHANGE-COMMAND-EXIT.  EXIT.

$PAGE "[40]  PROCESS DELETE COMMAND"
    40-PROCESS-DELETE-COMMAND       SECTION.

        DISPLAY "PROCESSING FOR THE DELETE COMMAND".

    40-PROCESS-DELETE-COMMAND-EXIT.  EXIT.

$PAGE "[95]  FINISH UP"
    95-FINISH-UP                    SECTION.

        MOVE "CLOSE"                TO QHELP-COMMAND.
        CALL "QHELP" USING QHELP-AREA.
        IF NOT QHELP-OK THEN
            DISPLAY "ERROR:  UNABLE TO TERMINATE QHELP SYSTEM".

    95-FINISH-UP-EXIT.  EXIT.
$PAGE "[99]  FATAL ERROR"
    99-FATAL-ERROR                  SECTION.

        DISPLAY " ".
        DISPLAY "ERROR:  FATAL TERMINATION OF THE PROGRAM EXAMPLE".

        STOP RUN.

    99-FATAL-ERROR-EXIT.  EXIT.
```

This example does not cover all of the QHELP commands available to the COBOL programmer. It is possible, with the *FIND* and *UNFIND* commands, to position the internal QHELP pointers to various levels of the help tree. These commands could be used to position the HELP file to the HELP text associated with the add command, when processing the add command in the example above.

There is also a QHELP set command, which allows various QHELP defaults to be overridden. These include options such as the number of lines on a screen, and the type of pattern matching that QHELP should use when looking for keywords.

ADVANCED QHELP FILE STRUCTURE

The basic file organization of QHELP files is one file with many \BEGINKEY and \ENDKEY commands, where \BEGINKEY commands can be nested up to a level of 10. For very large systems this is too cumbersome. It is necessary to have separate HELP files for each major system module.

QHELP permits a single "logical" HELP file to be separated into several physical files (which can also act as stand-alone "logical" files). This is done by including an entire \BEGINKEY - \ENDKEY pair in one file. In the QEDIT example above, assume that all of the information having to do with the set command was to end up in a file called SET.HELP.ROBELLE. The primary QEDIT HELP file would then include the following entry:

```
\BEGINKEY SET [SET.HELP.ROBELLE]
\ENDKEY SET
```

Any key can be in another file, but nesting of files is only permitted to five levels. The [>]. indicates that the key is located in the file specified inside the [>]. The file SET.HELP.ROBELLE is the file which contains all of the HELP information about the QEDIT SET command. This file is a regular QHELP file, which starts with \BEGINKEY SET and ends with \ENDKEY SET.

The layout of the SET file would look like this:

```
\BEGINKEY SET

    ...  General information on what the /SET command does.

\BEGINKEY GENERAL


    ...  This key belongs to the SET command.  It describes all
         parts of the set command except /SET OPTION.
```

```
\ENDKEY GENERAL
\BEGINKEY OPTION
     ...  Each of the /SET OPTION parameters is described here.
\ENDKEY OPTION
\ENDKEY SET
```

The SET HELP file can also be used as an individual HELP file, in addition to acting as the entry for the SET key in the QEDIT HELP file.

Modular Programming in MPE

Ingenieurbüro Jörg Grössler
IJG, Gbgh, Berlin

MODULAR PROGRAMMING

— There is no final definition yet
— A module can be embedded into any environment knowing its interface but not the algorithm used.
example:

sin (×)

the user must know:
— x must be of type "REAL"
— sin (×) will be of type "REAL"
— sin (3.1415) = 0
— 1.2E−50 < × < 4.5E+55
— what happens in case of error

the user must not know:
— the method how sin (×) is calculated

SOME MORE ASPECTS

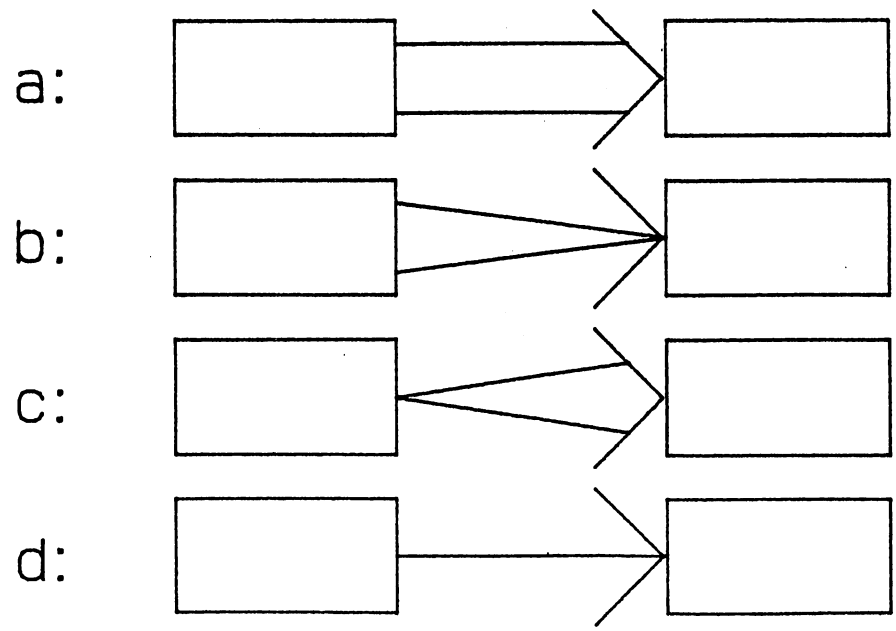
— A module can be constructed without knowing the environment it will be used in
— The module interfaces should be as simple as possible

WHAT MODULES CAN OFFER

— Procedures
e.g.: sin(×)
— Data
e.g.: INTEGER ARRAY A
— Files
e.g.: Data-base
— Any mixture of the three above

MODULE INTERFACES

— Information flow between modules
— Described by:
— The type of information (data, procedure, file)
— The access rights for each communication direction:



Examples for Module Interfaces

```
a:   BEGIN
       INTEGER I;
       ...
       PROCEDURE P1;
         BEGIN
           I:=0;
           WHILE (I:=I+1) < 10 DO
             BEGIN .... END;
         END;
       ...
       I: *EQL0;
       WHILE (I: =I+1) < 10 DO
         BEGIN
           ...
           P1;
         END;
       ...
     END;
c:         SUBROUTINE SUB
           ...
           INVAL=ITEMP (10)
           ...
           END
```

MODULE REQUIREMENTS

- — Control of information flow (specification of imported and exported objects)
- — Check of interfaces (some checking done by SEGMENTER, but not for all types)
- — Hidden information (to keep information within the module — problems with stack-structure, file-access)
- — More possibilities to restrict access on data, procedures and files
- — Comfortable to handle (library-problem)

Example: Own Data in SL-Routines

PROBLEM: The principle of hidden information requires that local data is not deleted between two procedure calls. This causes problems when procedure has to be put into a SL.

WHAT WE WANT: A module which stores local data into an extra data segment before exit and refreshes the data after call.

SPECIFICATION FOR MODULE "OWN DATA"

```
PROCEDURE INITDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;
  OPTION EXTERNAL;

  BEGIN
    IF `first time used'
      THEN `initialize BUFFER with 0'
      ELSE `refresh BUFFER with data
           stored in data segment';
  END;
```

```
PROCEDURE UPDATEDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;
  OPTION EXTERNAL;

  BEGIN
    `copy contents of BUFFER into
     data segment';
  END
```

Solution No. 1

```
PROCEDURE INITDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;

  BEGIN
    `allocate data segment';
    IF `data segment already exists'
      THEN `copy contents into BUFFER'
      ELSE `initialize BUFFER with 0';
  END;
```

```
PROCEDURE UPDATEDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;

  BEGIN
    `allocate data segment';
    `copy contents of BUFFER into
     data segment';
  END;
```

But

— Extra data segment has to be "global."

Therefore:

— Other users of module "OWN DATA" will use the same data segment

— Data segment is not automatically deallocated when program terminates. So no initialization will happen after the module has been used once.

Solution No. 2

```
PROCEDURE INITIALIZEDATE;
  OPTION PRELUDE;

  BEGIN
    `allocate extra data segment';
    `mark user within data segment';
    `initialize info part';
  END;
```

```
PROCEDURE INITDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;

  BEGIN
    `allocate extra data segment';
    IF `used first time (info part)'
       THEN
         BEGIN
           `initialize BUFFER with 0';
           `change info part';
         END
       ELSE `copy contents into BUFFER';
  END;


PROCEDURE UPDATEDATA (BUFFER, LENGTH);
  INTEGER ARRAY BUFFER;
  VALUE LENGTH; INTEGER LENGTH;

  BEGIN
    `allocate extra data segment';
    `copy contents of BUFFER into
    data segment';
  END;


PROCEDURE FREEDATA;
  OPTION POSTLUDE;

  BEGIN
    `allocate extra data segment';
    `delete module user from info
     part';
    `free extra data segment';
  END;
```

Business Graphics: An Efficient and Effective Tool for Management Decision Making

Gavin L. Ellzey
Systems Engineer
Hewlett-Packard
Kenner, Louisiana

ABSTRACT

Presently, one of the major dilemmas facing management is the communication of pertinent business information. In this paper we will address the subject of computer graphics as an efficient and effective solution to the problem of the "Information Explosion" faced by today's manager.

INTRODUCTION

Yosemite. To any of us who have been there, the name conjures up vivid recollections of the majesty of California's High Sierra. The stark, granite majesty of El Capitan rising over 3000 sheer, vertical feet from the floor of a pristine valley fed by the clear waters of the Merced River. Beautiful? Certainly, but for those who've seen the magnificence of the Yosemite Valley, or any other natural wonder, we realize that mere words fall far short of actually presenting a clear, concise mental image which does justice to the reality.

Although words have served admirably throughout man's history for the cataloging and storage of information in printed form, in today's complex and ever-changing business environment they frequently fall far short of presenting the full picture of the business event(s) they represent.

If you agree with this perception, then read on, for you are ready for the new age in computers, the "Graphics Age."

Graphics, as we know them, are not an invention of the twentieth century, or for that matter even the nineteenth century! In fact, graphics can trace its origins beck to 1786 when William Playfair first published *The Commercial and Political Atlas* in which he presented what he called "lineal arithmetic,"[1] the progenitor of modern graphics.

Being practical, as most businessmen are, we may now ask ourselves, "Good, but how do graphics relate to me and my business?" The answers to that question lie in the following pages.

Presently, the computer industry is in a perpetual state of flux as new technological advances allow the costs of computer hardware to steadily decline, bringing computerization well within the budgets of even the smallest businesses.

The old adage, "The more you got, the more you want," still holds true; as the data storage cost of computers decreases and capacity increases, the volume of "neccessary" data increases also, thus giving rise to a major dilemma facing today's manager, the "Information Explosion."

Today's manager is essentially the same as his counterpart of fifty years ago, a decision maker. However, the complexity of the present business environment is substantially greater and increasing daily!

Managers, being decision makers, are under extreme pressure to assimilate information, delineate that information which is useful and use it to make crucial business decisions as quickly as possible.

Keying on this we see that the most valuable commodity to today's manager is time. With the advent of computers and the birth of the "Information Explosion," managers are finding themselves buried under mountains of computer printouts containing the vital statistics of their businesses. Studies have shown that the average human can read at 600 to 1200 words per minute, with exceptions at both extremes.[4] Combining this with the time it takes the brain to correlate and interpret the information read, we can see that it could take quite a while to delineate the trends contained in a 50 page line printer listing of business statistics.

Today, however, the old man-computer interface channel of the line printer listing can be rendered much more effective by being augmented with computer graphics. Given our previous example of a 50 page printout, the information can be mapped onto one well designed graph which only takes up a single page! Furthermore, when presented graphically trends can be spotted immediately, thus greatly minimizing the decision making time. Thus, where it once may have taken a manager two hours to make sense out of a printout, he can now do it at a glance!

It can now be seen that computer graphics provides managers with two fundamental benefits. Primarily, it

saves on the one resource always in short supply to all managers — time.

Secondly, computer graphics greatly enhances the effectiveness of the decision making process by making pertinent information easier to understand and trends easier to recognize.

USING BUSINESS GRAPHICS

There exist, primarily, two types of graphics reporting categories. First there are the special one-time uses such as forecasting, customer presentations and specific problem analyses, to cite some examples. Second, there are the management reporting functions. These provide a periodic set of charts which present a clear "state of the business" picture to the managers who use them.

Having defined the two primary areas of chart use, we must now ask ourselves, "How do I begin?" There are some preliminary criteria which must be satisfied by any enterprise attempting to implement a business computer graphics system. Familiarity with the data which forms the building blocks of the system is paramount, also the designer must choose specific quantities for measurement which correspond to definite measureable goals.

An easier way to approach it may be from the standpoint of the "5-Ws and H;" Who, What, Where, When, Why and How.

- WHO — Who will the intended audience be, an executive vice-president or a shop foreman.
- WHAT — What quantities are going to be measured, cash flow or inventory on hand.
- WHERE — What area of my business am I interested in, where among some or all of my divisions are my questions centered.
- WHEN — When, what time or associated interval do I want to survey; do I want a monthly comparison or just a daily trend analysis for a given month.
- WHY — For what reason do I want to investigate these quantities and their various relationships, waht are my fundamental goals and what benefits will be derived.
- HOW — How will the data be presented, in linear form, pie or bar chart.
- These questions must be carefully and intelligently resolved by a joint effort between Management and the DP Staff prior to any graphics system implementation. You must know where you want to go in order to choose the right path and proper means of transportation.

Once the objectives have been set and the method of accomplishment determined, the next step is the design of what is called a "Graphics Operations Portfolio."

A Graphics Portfolio is simply a predetermined set of charts generated on a regular basis to provide management with a clear and concise overview of the business. It should include not only charts, but their supporting text and tabular data arranged such that they complement each other and are optimized for maximum effectiveness. It should be remembered during the entire process that there are some forms of data which do not lend themselves to graphical presentation; those where the absolute value of the quantity is the important criterion. Anyone can see that finding appropriate answers to the "5-Ws" is not too difficult for any manager well versed in his business. Experience has shown that answering the "How" is where most people begin to lose themselves, therefore we will now explore the various comparisons and their associated chart types.

There are five major types of comparison which may be used to define most data sets:

1. COMPONENT — compare relative magnitude of a particular segment in relation to the whole.
2. ITEM — compare the ranking of individual items.
3. TIME relative & absolute — compare the variation over a given time interval of an item(s).
4. FREQUENCY DISTRIBUTION — compare how a quantity is distributed among various categories.
5. CO-RELATIONSHIP — present a view of how an item varies in relation to variation in another item.

In general, most management reporting functions can be fulfilled by the three most fundamental types of charts — pie, bar (normal, stacked, comparative) and linear. Ecah chart type functions as a means to depict a specific type of comparison thus serving a different purpose when an element of the Graphics Portfolio.

We will now examine the various comparisons and their associated chart types.

Component

Component comparisons show the relative importance of any particular component in relation to the whole. There are two basic types of chart which lend themselves effectively to the representation of component comparisons: pie charts and stacked bar charts.

PIE CHARTS — Many mathematicians consider the circle the "perfect" geometric figure, and if we look at the anatomy of a circular or "pie" chart we can see that since progression flows continuously from start to finish, a pie provides us with an exceptionally vivid impression of a "whole," making it an ideal figure for the representation of precentage type comparisons. An important rule to remember when designing a pie chart is to use as few "slices" as neccessary, preferably less than 7.

STACKED (100%BAR/COLUMN) BAR CHARTS: The stacked bar provides us with the "skyscraper" effect, with the various segments being the "floors." This is a great type of chart to show numerical component comparions among multiple components.

A question now arises as to when a pie or a stacked bar chart should be used. Generally, due to the human tendency to think of things circular as "complete" a pie chart gives a more concise impression of a whole, fur-

thermore simple experimentation for ourselves will show us that the relative sizes of component segments are easier to determine accurately when arranged in a circular arrangement as opposed to a "cubed" stack of variable sized cubes, thus making the pie chart ideal for presenting the components of a single total. However, sometimes we have the need to show the components of several totals simultaneously and for this the stacked bar chart is the ideal chart since multiple component comparisons can be placed adjacent to each other more easily than the pie chart. For example take the following data:

1980 XYZ COMPANY SALES

DIVISION	SALES $	QTR1	QTR2	QTR3	QTR4
A	100	30	30	20	20
B	300	101	59	40	100
C	250	50	35	65	100

Looking at Figure 1 we see that the pie chart provides a better measure of the relative magnitude of the respective sales divisions. Figure 2 depicts a component comparison on a quarter by quarter basis by division. It can be easily seen that had we tried to use pie charts for this it would have taken four charts and been rather unwieldy, but more importantly not as effective.

Item Comparison

The item comparison treats two or more items and compares their relative sizes or quantities to each other. Bar charts, both horizontal and vertical, are the chart types best suited to the representation of this type of comparison. Much discussion has taken place concerning the relative advantages/disadvantages of using the vertical or horizontal bar chart over its counterpart. The primary argument against vertical bar charts is that some people tend to read in a time dependant relationship. It is for the chart designer to decide which type of bar chart to use in order to best suit the need of his audience. I have included examples of both in Figures 3 & 3A. We will now look at an example. Take the following data:

ABC COMPANY
FISCAL 1980 SALES

DIVISION	SALES $
A	342
B	109
C	226
D	374

If we look at Figure 3 & 3A we can see how this item comparison maps into both a horizontal and a vertical bar chart.

Time Comparison

Time comparison shows how a quantity fluctuates over a specified time interval; relative or absolute. The primary goal of a time comparison is to emphasize trends and their associated patterns by showing fluctuations in a quantity's value. Fundamentally, two types of charts are suited to the presentation of time comparisons: linear charts and bar charts (normal & comparative).

BAR CHART — The anatomy of a bar chart is such that when we are dealing with a minimal or small number of time periods, such as a 12 month analysis, then it is ideally suited for our presentation vehicle. Furthermore if we wish to compare several components per interval, the comparative bar chart is an excellent choice. The reason we do not wish to use bar charts for comparisons involving many time intervals is that after a certain point, the chart becomes cluttered with too many bars. Also, due to the visual perception of figures versus lines, if there exist large fluctuations is the data then the mental conceptualization of the trend is superior when it is represented by a bar chart than if a linear chart is used.

LINEAR CHART — Linear charts are usually drawn on Cartesian or Logarithmic axes. Due to the nature of these charts, a linear chart lends itself well to representing large numbers of data points.

Let us look at an example based on the following data:

KLN WIDGETS, INC.

YEAR	SALES$	QTR1	QTR2	QTR3	QTR4
1971	301	100	60	40	100
1972	400	75	95	100	130
1973	411	80	71	150	100
1974	401	200	100	70	30
1975	511	100	20	300	80
1976	494	100	204	100	90
1977	509	75	125	159	150
1978	564	264	100	110	90
1979	580	280	100	200	80
1980	600	174	176	100	150

In Figure 4 we see that since we are looking at only 5 discrete time intervals, then the bar chart is superior in representing magnitude of the item fluctuation, over the linear chart in Figure 4A. Figure 5 gives us an example of how a comparative bar chart can be used to show the fluctuation of several items over several time intervals. Looking at Figure 6, we can see that this time series comparison of a quarter by quarter sales record over ten years is ideally suited to a linear chart representation. Had we tried to depict these trends in a bar chart two things would have occurred: the bars would have been miniscule and the chart would have been so cluttered as to be quite difficult to effectively interpret.

Frequency Distribution

This type of comparison shows how the given quantity fluctuates over a given distribution. The key to this is that broad categories are used to define a distribution over which the data values are dispersed, rather than specific events. Two types of chart are suitable for representing this comparison, bar charts or linear charts. As with the tine frequency comparison, when many points need to be graphed the linear chart is most effective. For a small number of categories, the bar chart is the most effective format. Bar charts used in this manner are commonly called Histograms. Figure 7 shows a sample histogram for the following data:

KLO WELDING
1980 EMPLOYEE ILLNESS

YEARS SERVICE	DAYS ILL
0 - 55	50
5 - 10	150
10 - 15	75
15 +	40

Co-relationship Comparison

This type of comparison functions to highlight pertinent patterns in the relative fluctuation of one item with respect to another. Scattergrams — linear charts with figures representing each data point — are the major chart type for representation of these types of comparisons.

SUMMARY

Summing up, we have seen that graphics, when properly understood and implemented can provide a powerful tool for a manager. Choosing the proper chart format and comparison for the relationship desired involves careful forethought and planning; just as a well designed Graphics Portfolio is a valuable managerial asset, a poorly designed one is almost useless. In closing, we can see that graphics, if properly implemented, greatly increases the efficiency and effectiveness of the decision making process.

BIBLIOGRAPHY

[1]Schmid, Calvin F. and Stanton E., *Handbook of Graphic Presentation.*

[2]*Business Week,* June 1980.

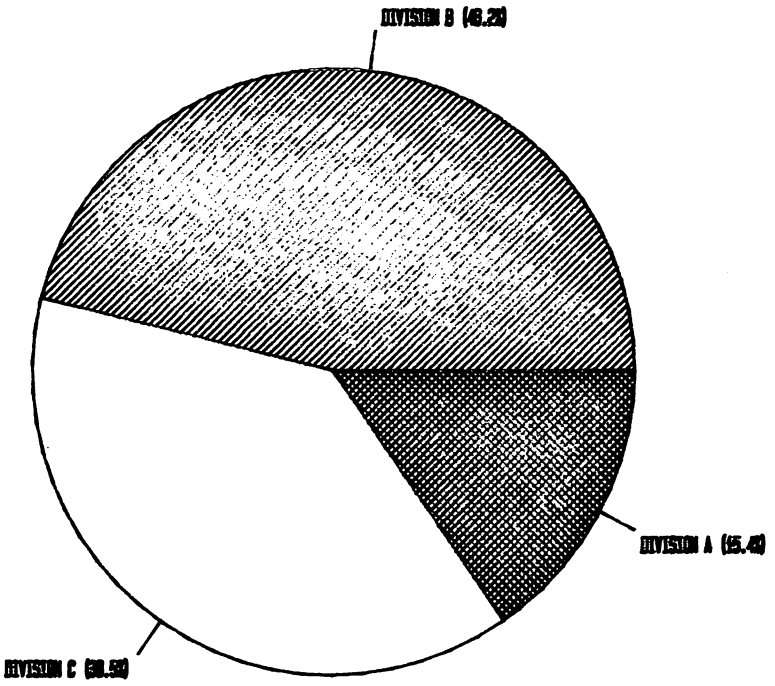
[3]Cook, Peter G., *25 Years of Computer Graphics,* IGC Conference for MIS and Computer Graphics, February 1978.

[4]Patterson, Marvin, *Graphic Representation of Numeric Data.*

ACKNOWLEDGEMENT

The author would like to express his appreciation to the following people for their invaluable assistance in locating sources of information for this paper: Bruce Woolpert & Peggy Wyman: HP San Diego, and Rich Simms, HP General Systems Division.

1980 TOTAL SALES FOR XYZ COMPANY



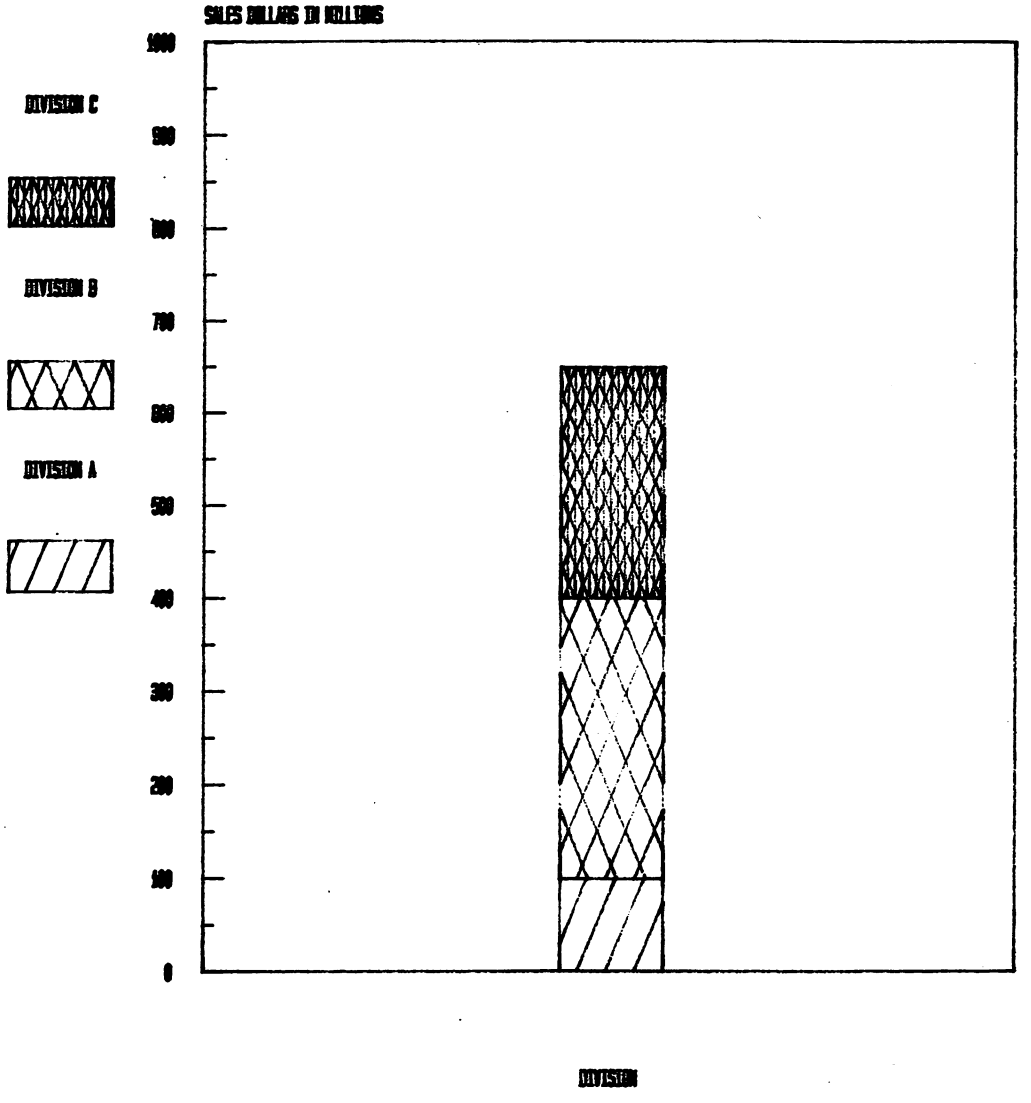


Figure 1

1980 TOTAL SALES FOR XYZ COMPANY BY QUARTER

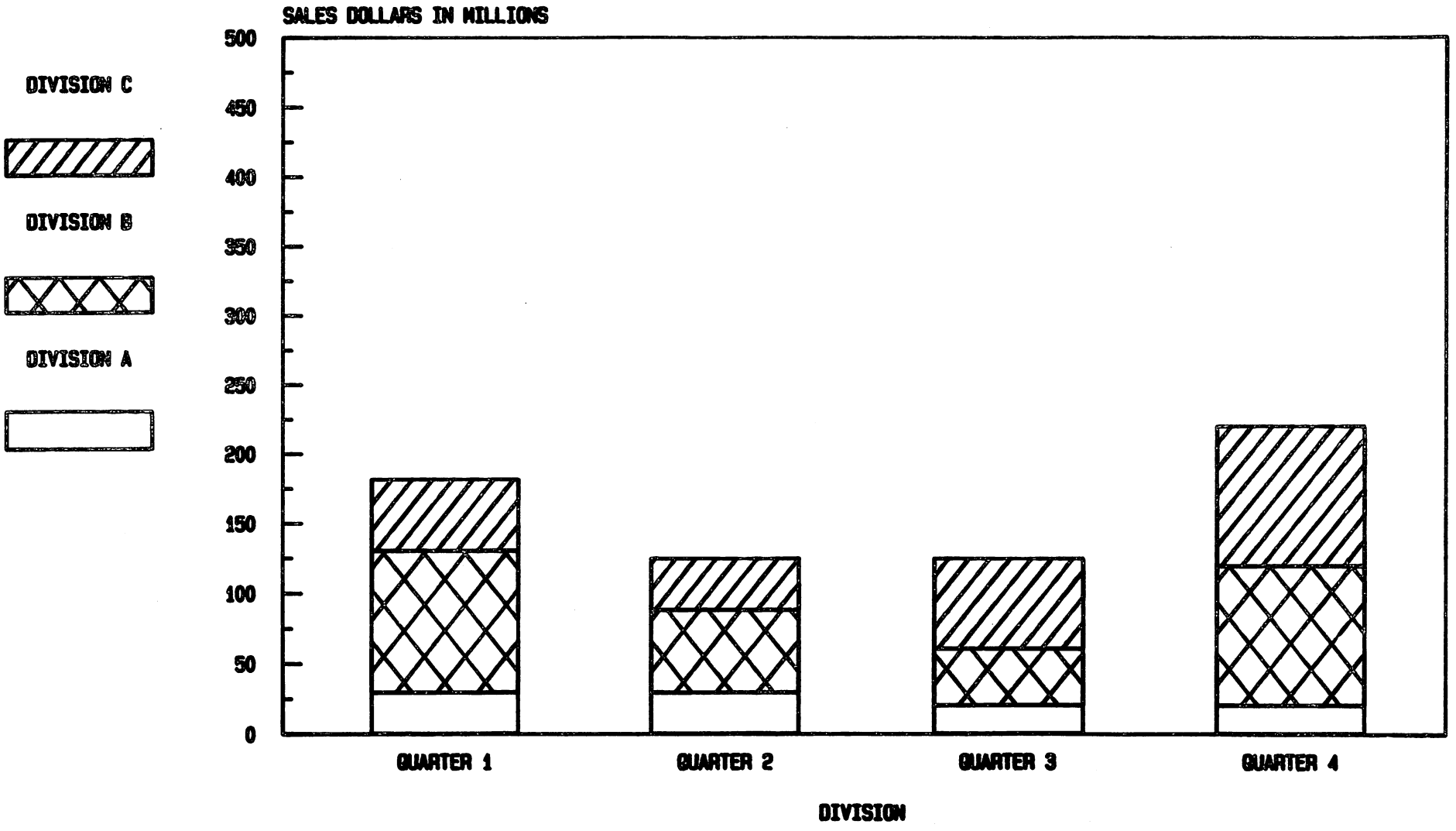


Figure 2

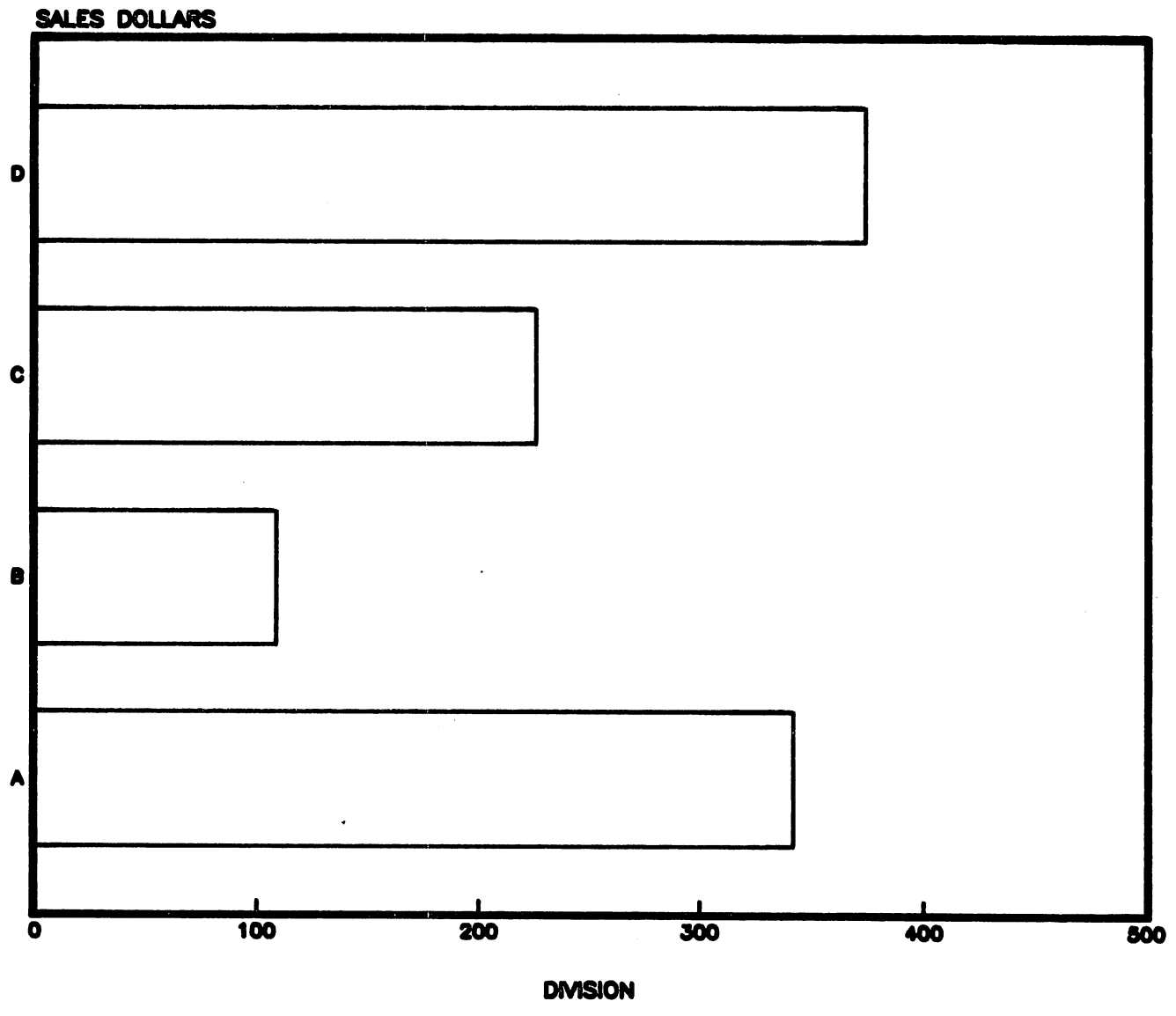

FISCAL 1980 SALES FOR ABC COMPANY BY DIVISION
SALES DOLLARS
D
C
B
A
0
100
200
300
400
500
DIVISION

Figure 3

FISCAL 1980 SALES FOR ABC COMPANY BY DIVISION

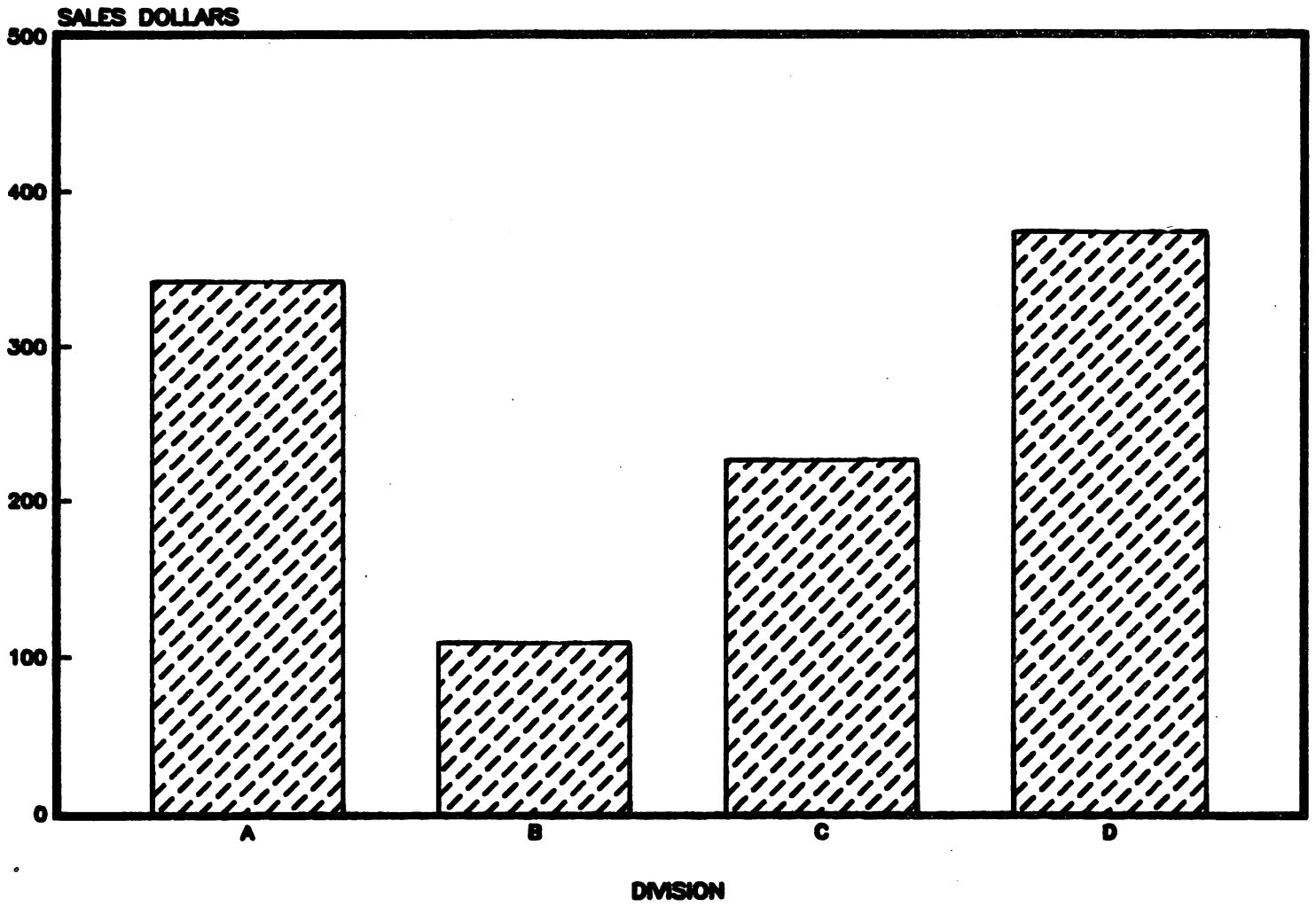


Figure 3A

KLN WIDGETS, INC. 5 YEAR SALES ANALYSIS

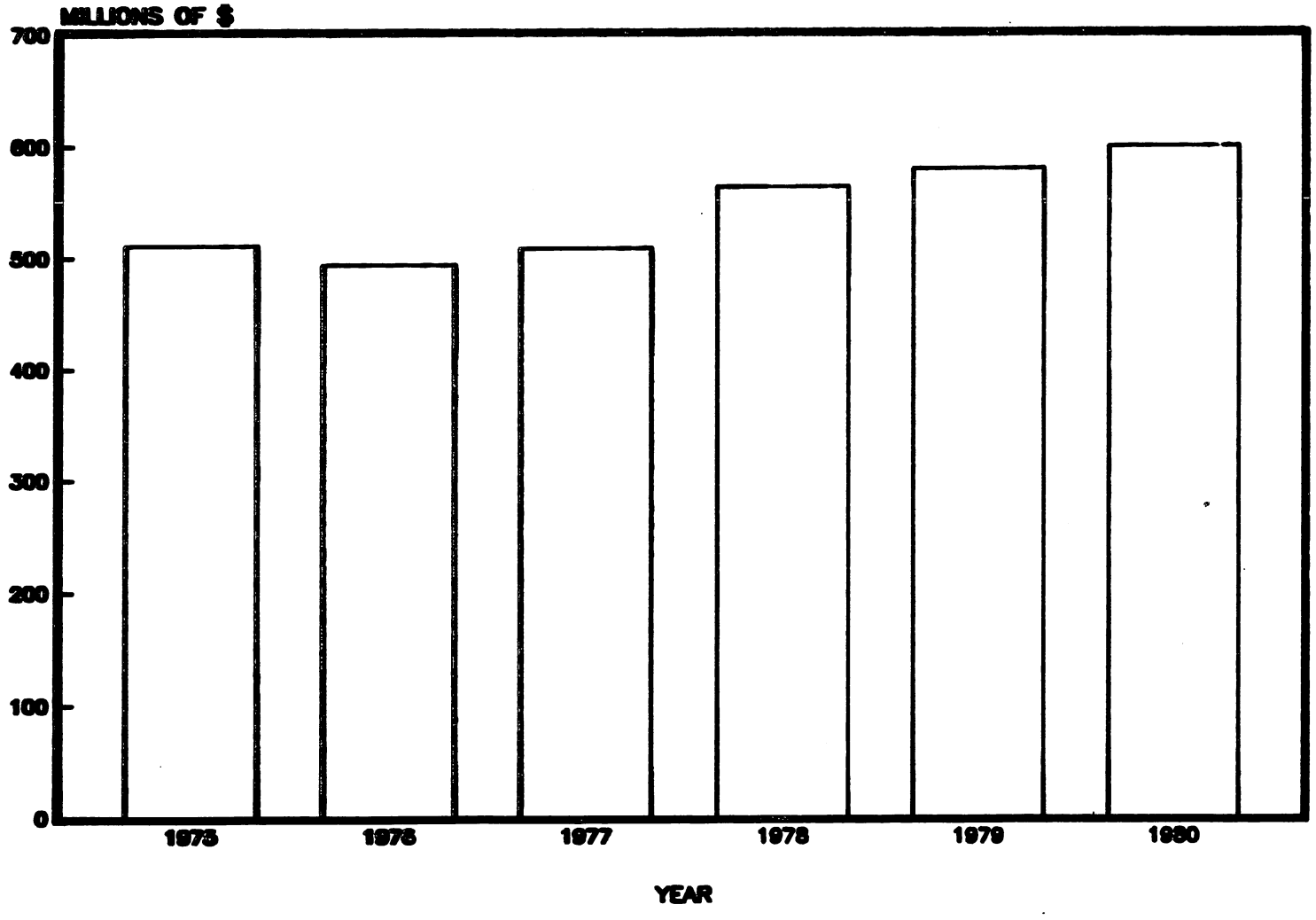


Figure 4

KLN WIDGETS 5–YEAR SALES ANALYSIS

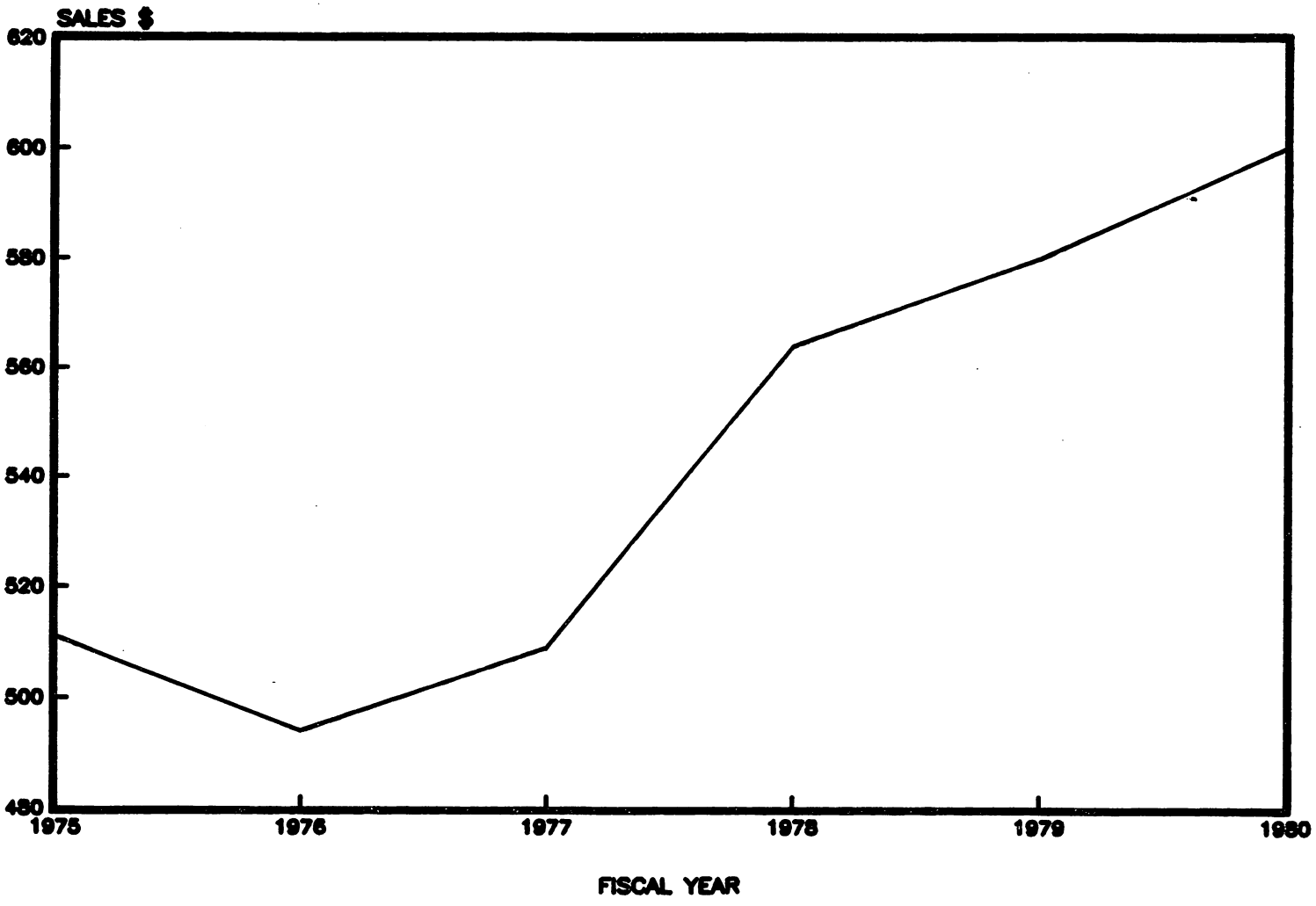


Figure 4A

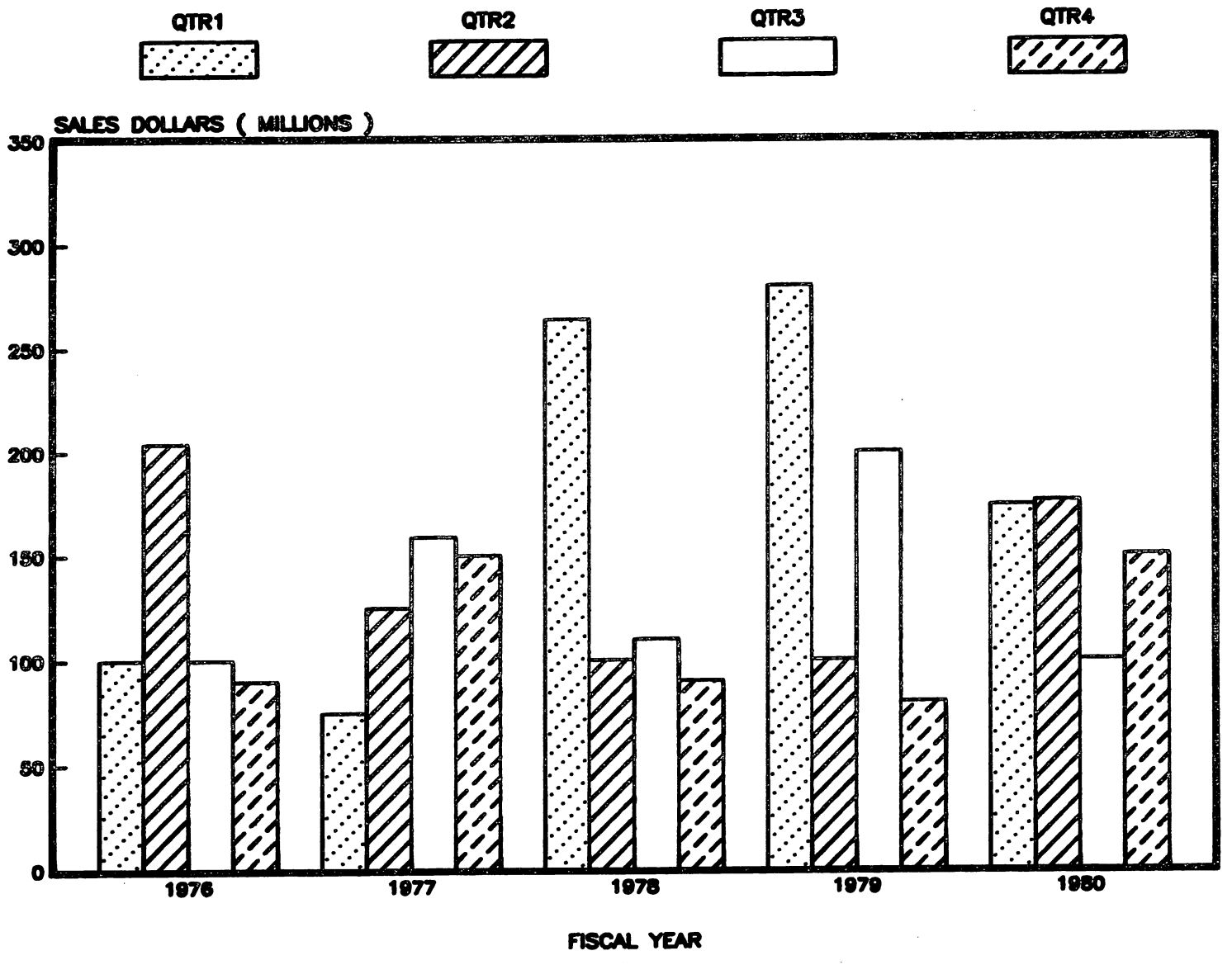

KLN WIDGETS,INC. 5 YEAR QUARTERLY ANALYSIS
QTR1
QTR2
QTR3
QTR4
SALES DOLLARS (MILLIONS)
350
300
250
200
150
100
50
0
1976
1977
1978
1979
1980
FISCAL YEAR

Figure 5

10 YEAR FISCAL QUARTERLY ANALYSIS

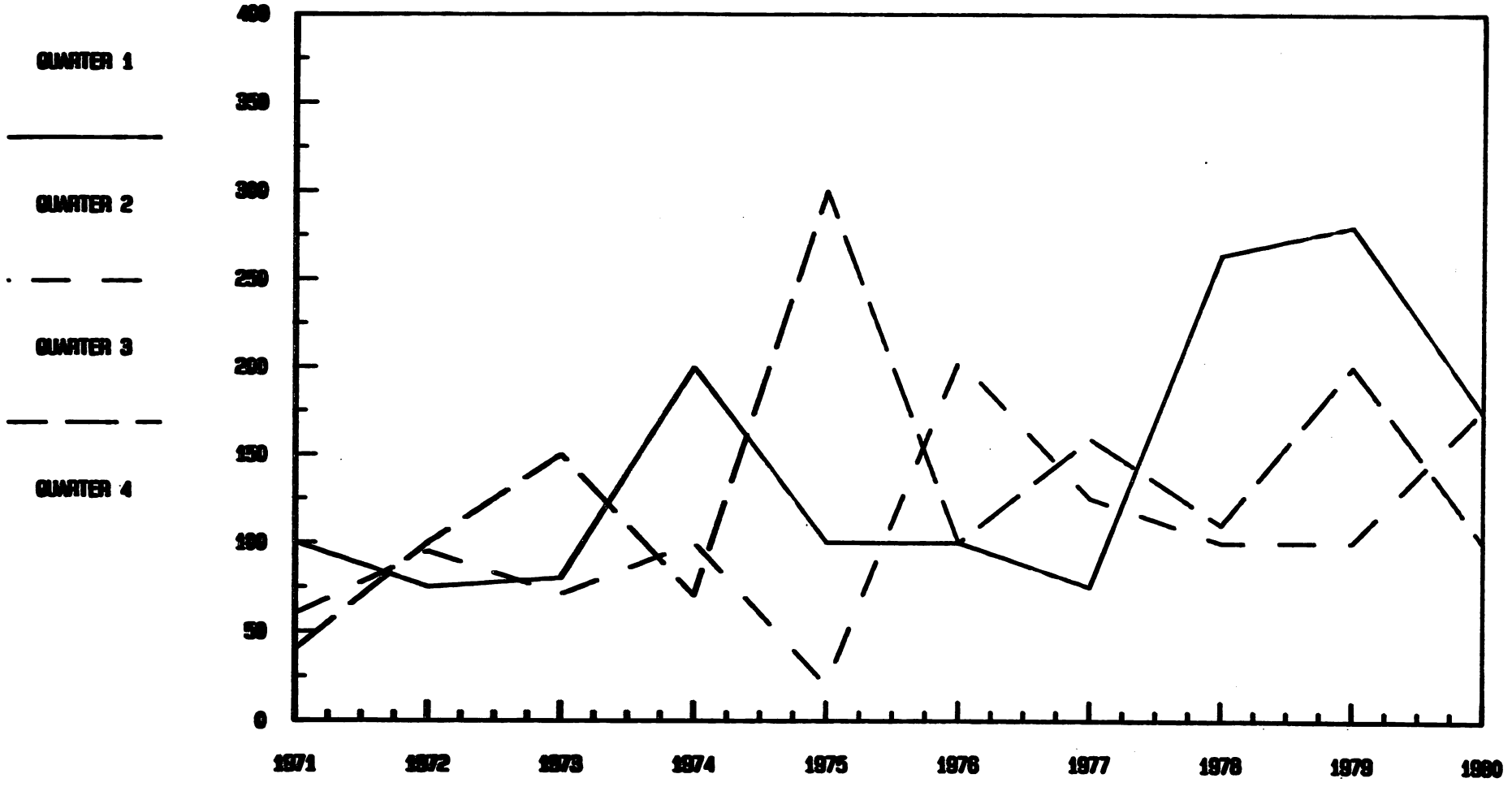


Figure 6

KLO WELDING : EMPLOYEE ILLNESS DISTRIBUTION

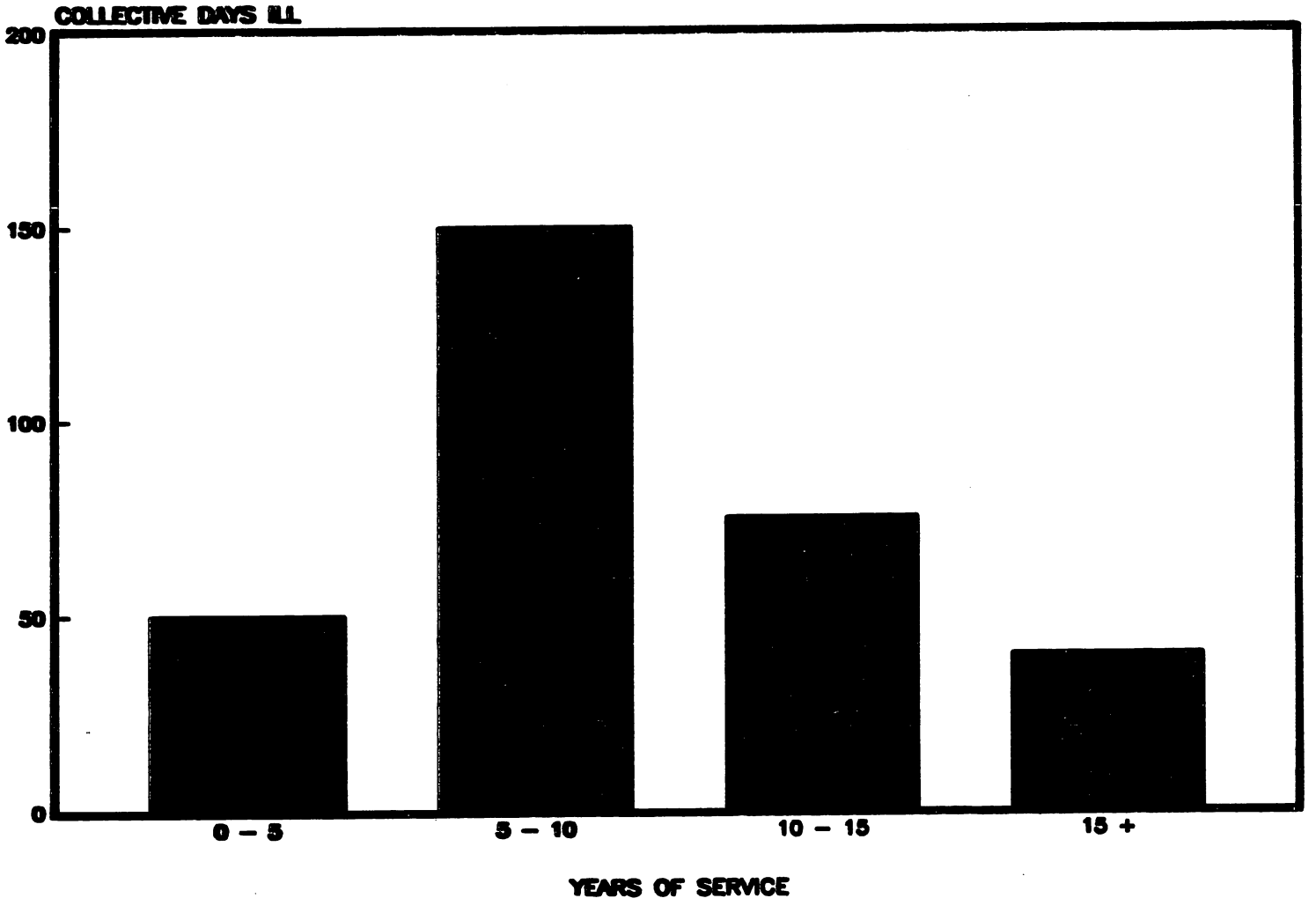


Figure 7

Automatic Calling with the HP3000

Paul W. Ridgway
Systems Engineer
Hewlett-Packard Company
Houston, Texas

INTRODUCTORY SUMMARY

The purpose of this paper is to introduce HP3000 users to the concepts and benefits of computer controlled call origination. Accurate information management is one of the foundations for a successful business and as businesses grow geographically, accurate information management between businesses becomes increasingly important.

The topic of information management between businesses has been addressed through the communications network concept with each vendor supplying their own competitive network design. In this paper, I will discuss methods of automating the existing communications network capabilities of the HP3000 and will propose new ideas of automating the communication of electronic information.

The body of the paper will consist of two sections. The first will be a technical discussion aimed at the non-technical individual. It will describe the components involved and how they interface the HP3000 with the outside world. Also, the first section will examine the software required and will give examples of how to integrate the software into existing network modules. The second section will deal with the concepts of automatic call origination and will include a discussion on the need for automatic information transfer and a summary of existing and proposed appplications which are suited to the HP3000.

To conclude, we will work out the economics of automatic call origination and compare its advantages to other methods of networking.

INTRODUCTION

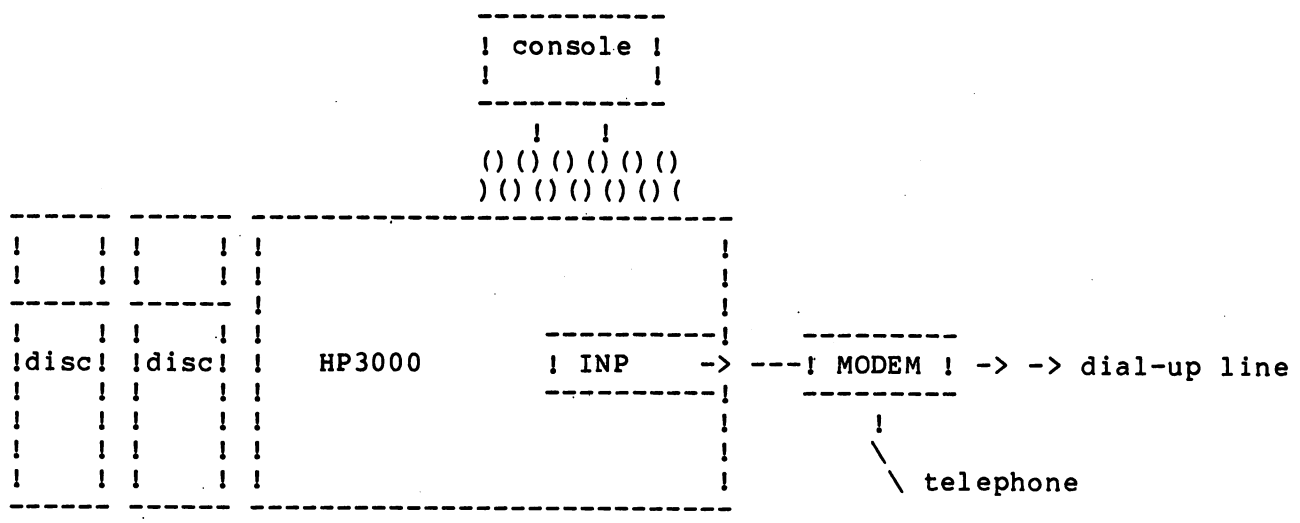
I am an observer of the Cosmos to use Dr. Carl Sagans' word to mean that I get input for my thinking from every source of information known to man. For example, if I want information about the universe, I select a source of that information, say a book, and read. If I want information about the state of politics overseas, I select a television channel that is displaying that information and watch. In these cases, I know what *kind* of information I want, I know *where* to get it and I have the ability to make contact with the source of information through my actions.

This seeking out, absorbing and updating of data is called information management and is an integral part of human existance as well as of modern business. For example, a company with employees that are paid weekly has a data entry method for giving the computer information about the number of hours worked, sick leave and vacation earned or taken, pay rate increases and so forth. Once each Friday, the payroll program runs, extracts the information and produces output in the form of paychecks. For a single office business with few employees, (and small quantities of information) this scheme works fine. However, if the business has a home office and several branch offices across the continent, the communication of information becomes a formidable and costly overhead operation.

This paper will address two facets of information communication: the ability to select the information source or destination and the timliness of communication with regard to cost effectiveness. To clarify these facets: automatic selection of source or destination is analogous to me as an observer; I can select a book or television as the source, depending on what I need to know. The timliness of communication is analogous to the idea that I can select the source or destination when I need it and in the case of the home office — branch office example, when it costs less to transmit or receive the information.

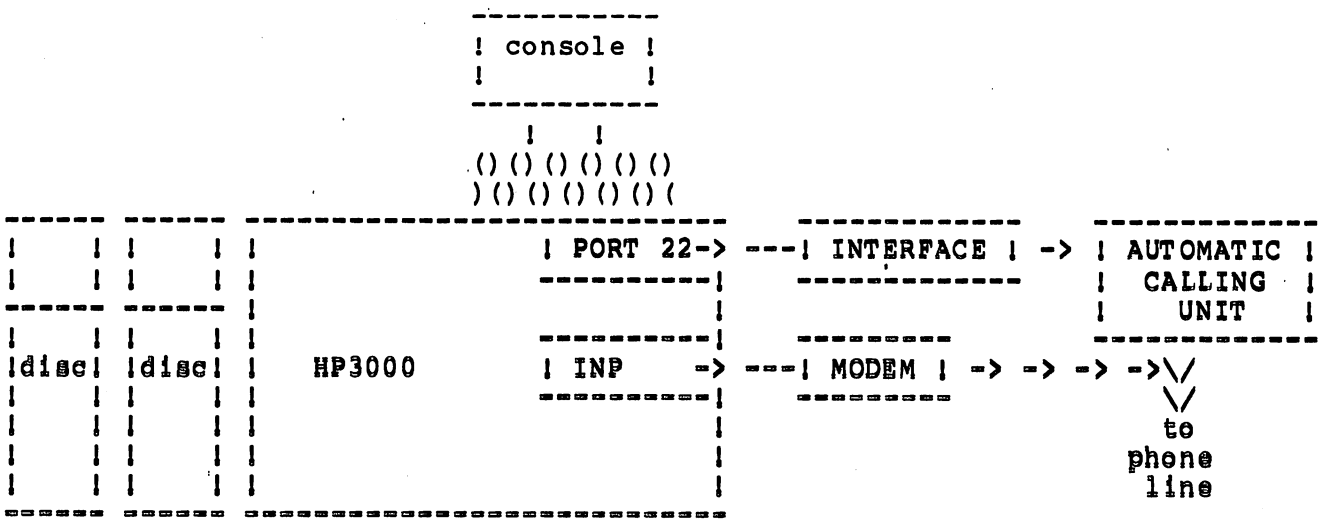
HARDWARE and SOFTWARE

Let's now take a look at the first part of the discussion, the autodialer hardware. Consider this typical environment:



Using HP's Distributed Systems software and the Intelligent Network Processor (INP), a user of this system can manually place a call to another HP3000 of similar configuration having auto-answer capability and transfer data in either direction by issuing the right commands. Neither system has any special capability and call origination is done manually by the operator.

To apply an autodialer to the system, the diagram would look like this:



The additional hardware is the Autodialer Interface and the Automatic Calling Unit (ACU). Within the ACU is a special mechanical switch which determines whether it or the modem is connected to the dial-up line. This switch has two states: "ready" and "busy." When the line is not being used the switch is in the "ready" state and connects the phone line to the ACU. When a communications link has been established with another system, the switch is in the "busy" state and the modem is connected to the line.

The effect of the autodialer (consisting of the interface and ACU) is to remove the manual operation of picking up the phone, dialing and waiting for a modem to answer and placing the data set online. Control and monitoring of the autodialer is done entirely through software which sends and receives control codes through the asynchronous port 22. Note that the actual data exchange between the two computers has not changed; existing DS software is used as before.

The ACU is the workhorse in that it is responsible for siezing the line, acknowledging a valid dial tone, sending a sequence of tone pairs down the line (the phone number) and recognizing the response. Typical responses from dialing a number are: no response, no answer, busy signal, human voice answers or a valid modem answers. In the case of a modem answering the line, the ACU can tell what type of modem responded by the carrier frequency placed on the line by the modem. The ACUs job is to tell the interface what response occurred. If a modem of the right type answers, the ACU must also flip its internal switch transferring

control to the local modem. The interface and ACU communicate with each other using a protocol called RS-366, the paralell equivalent to the familiar serial RS-232C standard. Thus, the interfaces' job is to translate RS-366 to RS-232C ASCII and vice versa such that the HP3000 can control the ACU. The HP3000 software does this by sending a string of ASCII characters to the port and reading the port for the response as though it were a simple file.

For a look now at the software side, this is a skeletal flowchart for a procedure that would connect any two systems together:

```
PROCEDURE DIAL ( PHONE'NUMBER , RETURN'CODE );

        START
          !
          !
    READ INTERFACE STATUS
          !
     IS LINE OCCUPIED?  YES -> RETURN'CODE := 1 -> RETURN;    (ERROR)
          ! NO
          !
     SEND A PHONE NUMBER
          !
          !
    READ INTERFACE STATUS
          !
       STATUS = "A" ?    YES -> RETURN'CODE := 0 -> RETURN;    (NO ERROR)
          !                                                   (MODEM ANSWERED)
          !
       STATUS = "B" ?    YES -> RETURN;CODE := 2 -> RETURN;    (ERROR)
          !                                                   (BUSY NUMBER)
          !
       STATUS = "C" ?    YES -> RETURN'CODE := 3 -> RETURN;    (ERROR)
          !                                                (INCOMPLETE DIAL)
          !
HAS 30 SECONDS ELAPSED?  YES -> RETURN'CODE := 4 -> RETURN;    (ERROR)
          !                                                   (NO RESPONSE)
          !
    END PROCEDURE DIAL
```

A program containing higher level software would see the autodial process as a simple procedure or intrinsic call to the system and would pass and receive data through a parameter list. In the case of any error response, the calling program would have to decide what to do next; possibly retry the call or give the user a choice of actions. If the "A"' response is given, the program would proceed by logging on the remote system and commencing a data transfer operation. The flowchart for a "driver" program would appear as follows:

```
      PROGRAM 'DRIVER'

          START
            !
            !
      DIAL REMOTE MODEM
            !
            !
GET A LOGICAL DS LINE NUMBER
            !
            !
        LOG ON THE
      REMOTE SYSTEM
            !
            !
```

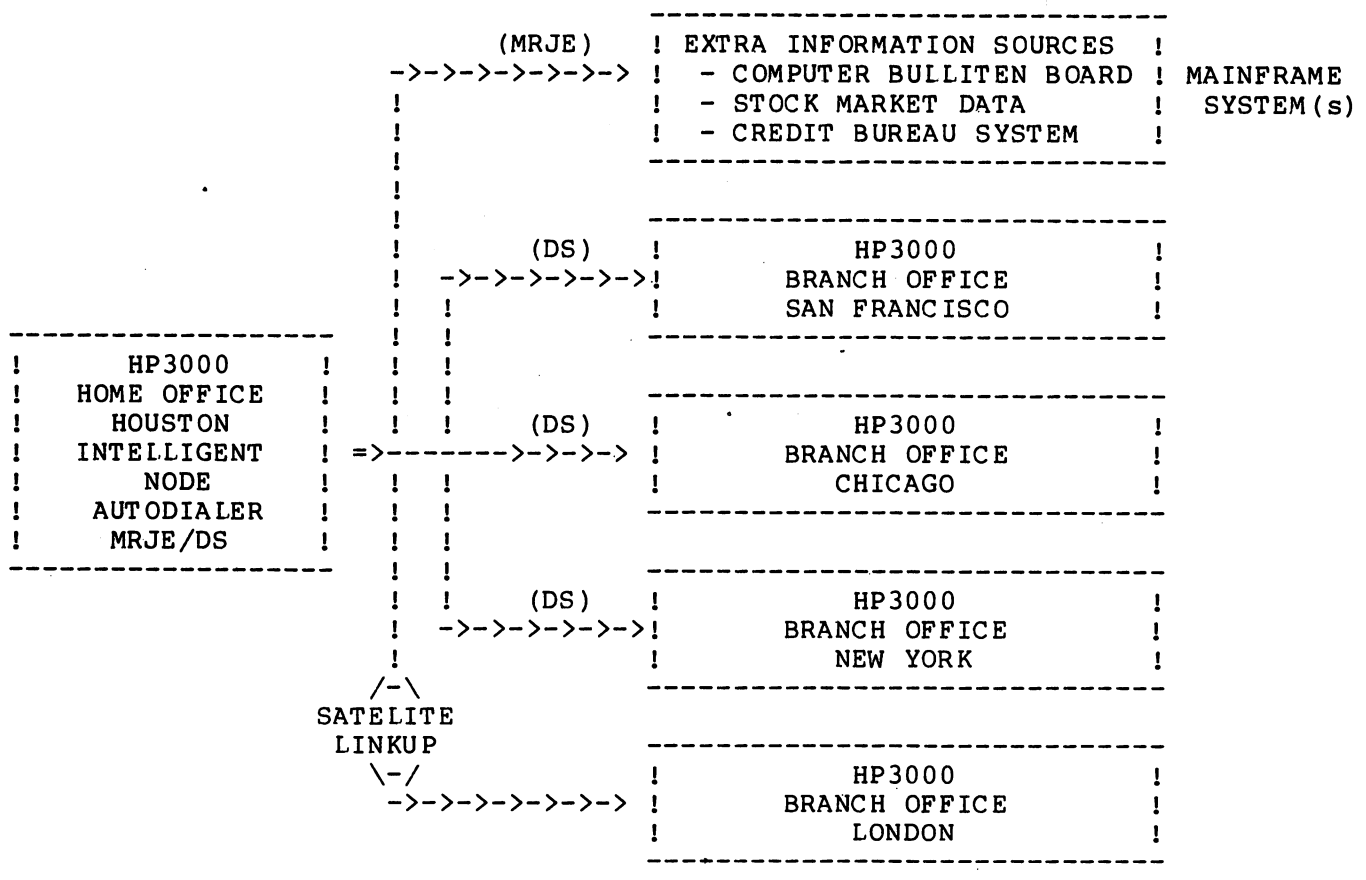
```
    SCAN THE 'AGENDA'
      FILE ON BOTH
        SYSTEMS
          !
          !
    PERFORM INDICATED
     DATA TRANSFERS
          !
          !
      LOG OFF THE
     REMOTE SYSTEM
          !
          !
CLOSE THE DS LINE
          !
          !
END PROGRAM 'DRIVER'
```

Within the driver program, the actions of scanning the agenda files and performing the indicated operations steps would have been previously set up according to the application using the autodialer.

CONCEPTS

This part of the discussion will deal with the concepts of automatic calling. The autodialer may be viewed as a special "hook" into a network of computers. The term "network" can apply to both existing connections between systems as well as *possible* connections between systems. In any case, the ability to share and thus manage electronic information lies in networking theory. To illustrate, let me present a quote from a popular small systems journal: "Some people forsee electronic information as the currency of the future: those who have it will use it to get more and those who don't have it will be exploited. Actually, money will probably continue to be the currency for years to come, but the computer will be the primary tool for controlling its flow. The key to this flow lies in computer networks. With the price of individual computers dropping, more businesses are solving their problems with distributed processing of computer networks rather than with a single large computer."[1]

If we accept this concept of dependence on the network, automatic access to the network is certainly a step in the positive direction. Giving a single system the capability of automatic access creates what I will call an 'Intelligent Node', one that can make unattended decisions based on access to the network. The intelligent node is actually a "smart" piece of software consisting of several logical modules, each having a specific responsibility. One such module is the autodial process. Another would be the one that logs on the remote system and another the one that handles file transfer and so on. A special feature of the intelligent node is that it is invisible to the network process. As an example, consider this diagram of a hypothetical warehousing and distribution business:



In this drawing, a business with distributed data processing is outlined. The home office is the "hub" or center of the network and has an INP configured for Distributed Systems and Multileaving Remote Job Entry use and an autodialer module. Surrounding the home office are the four branch offices each having a smaller HP3000 with DS and autoanswer modems. The London office is contacted through a commercially available satellite link such as Telnet and is accessed by the home office via the normal dial-up method.

Consider the following scenario:

Business function: Sales, warehousing and distribution of an industrial product or service which varies in cost, supply and demand on a daily basis.

Sales and product stocking are carried out at the branch offices while administrative and purchasing operations are handled by the home office. Data for pricing and sales volume need to be sent to and received from the branch offices on a daily basis to maintain the competitive edge. General electronic information about the stock market and credit information on the company's customers is stored on one or more systems which are not owned by the company but are used as extra information sources. The cost of the extra information depends on the demand based on the time of day; daytime access costs more than nighttime access.

Problem: The company is paying for an extra system operator whose only function is to manage the daily task of establishing contact with the remote systems and transferring the data to and from them. The cost of data transfer is high since all transactions take place between 8:00 am and 5:00 pm when the telephone rates are highest. The extra information sources charge high rates since access to them is made during prime time hours. The process is error prone since the operator follows a changing schedule which is often inaccurate.

Solution: The installation of an autodialer in the home office system removes the task of manually establishing contact with the remotes thus allowing the operator to perform a more valuable function for the company. Since all the systems are running 24 hours a day, data communications take place at night when phone rates are lowest and the extra information systems cost less to access. The process of contacting each system and performing the data exchange operations is done through a software "script"; a file of instructions for the intelligent node software to follow. Since the human element of error is removed, more accurate data transfers take place and the phone line use is optimized.

Operation: Every night beginning at 10:00 pm the Houston office automatically dials and connects with each remote system. The sales pricing and stocking data are exchanged with the remote systems such that when the next days work begins, every office has recent and accurate data. The extra information systems may or may not be contacted nightly depending on the need for that information. For example, if an employee in the London office wants credit information about an American client, she/he would put a request for that information in an "agenda" file on the London system. When Housto calls London that night, its software would scan the agenda file on the London system and receive the request for credit data. Then, the Houston system would contact the credit data system and extract the necessary information. When that transaction is complete, Houston would re-dial London and transmit the file to disc via DS software. Also, the software would make an entry to a log file documenting the exchange. The next day (or afternoon) the London employee would look at the log file, note that the data tranfer had taken place and find the name of the disc file containing the information she/he needs.

Results: The company has made more valuable use of an employee by removing a remedial task and assigning her/him a greater responsibility. The cost of communication has been reduced through the advantage of lower long distance rates at night. The extra information cost is also reduced since the non-owned systems are contacted during non-prime time hours. The company has increased its competitive edge since the remote systems always have accurate and up to date data concerning sales and stock.

The operational procedure of the intelligent node software running on the Houston system is the heart of the entire process and would be capable of dealing with any predefined situation that could occur. For instance, if any system being contacted did not respond or if a data transfer error occurred, the software would "know" what to do: warn the system manager, retry the transmission or attempt some corrective action.

Well, so much for a hypothetical example. Now I will describe two existing applications which use autodialers. The first is an HP3000 user who sells time on their system to clients for general business accounting use. One particular client is a local business that is a branch of a company that uses very large mainframe computer systems. The local branch office has only a terminal and communicates with the HP3000 with a 1200 baud acoustic coupler. In operation, the client maintains an IMAGE database containing data about the local branch activity. On an as-needed basis, the client will run a program which uses the autodialer to establish a dial-up link between the HP3000 and one of three mainframes located in distant cities. Then the MRJE/3000 subsystem is used to transmit the database to the host mainframe. Since the MRJE software is designed to automatically submit pending files and JCL whenever a data link is established, the process is very adaptable to the control of an autodialer. Without the autodial capability, the client would be limited as to when the mainframes could be accessed since an operator would have to be on duty at the HP3000 and would have to be trained in the operation of the dial-up link. This additional overhead would be an extra expense for the HP3000 owner and would be passed on to the client. Since the client is a small branch office, this additional, recurring cost would make the entire operation unfeasible.

The second example is an in-house application within Hewlett-Packard. This system uses autodial to communicate with other HP offices as well as with supported accounts having the DS capability. Software was written to allow a user of an in-house system to establish contact with other HP3000s on a named basis. For example, when I need to get a file from a system at our factory in Cupertino, I log onto my system and enter a single command with a parameter that is the "name" of the system desired. The command is actually a UDC which runs the software.

The program then finds the specified name in a database nd extracts the phone number of the remote modem and checks the user's security code. If the user attempting contact is not authorized to use the autodialer, he is denied access and an entry of the illegal attempt is logged to the database. This is a security measure designed to protect the remote systems by never allowing the users to know the phone numbers stored in the database. If the user is authorized to connect with the remote, the autodialer places the call and the link is established. Finally, the software gets a logical DS line number by calling the COMMAND intrinsic and prompts the user for a valid HELLO command. When the remote HELLO operation is successful, control is returned to the user. In all cases, the software logs the operation in the database, allowing us to measure the use of the communication facility. In the event of a user requesting access to an unknown system name, the program enters an heuristic mode where the user is prompted for the new phone number and a security code which defines what other HP3000 users may call the new remote system. The new name, number and security code are then verified for accuracy and entered in the database for future use.

CONCLUSIONS

As has been shown, the autodial hardware is capable of giving users of the HP3000 quick, easy and accurate access to other sources of information. With the cost of communication on the rise and the ability to get and manage electronic information becoming more crucial to todays business, the autodialer certainly has a niche in the data processing field.

One topic of concern is the economics of autodialing. How much does it cost? In the earlier days, about 5 years ago, Bell Laboratories introduced the model 801 automatic calling unit and leased it to qualified accounts for a monthly fee between $30 and $60. A Bell modem was also required and cost anywhere between $100 and $400 per month plus a healthy installation charge. In more recent times, with Bell removing the direct connection restriction to their phone lines, several vendors have introduced their own versions of datacomm equipment and Bell equivalent modems. One such vendor of autodial equipment is Racal-Vadic, offering an ACU for a one-time charge of about $750. Their ACU works with any modem they make and is ready for direct connection to the HP3000 and the phone line. Furthur, one ACU may be instructed to do the dialing for up to 16 modems of different types. With this ability, a local HP3000 using autodial to connect to the console port of a remote system through an asynchronous modem could in fact control the remote system completely - from performing file backup to actually shutting the system down and restarting it.

The concepts presented here are not new to the mainframe systems; they are new to the mini-computer users. This is due to recent quantum leaps in communications technology for minis and the increasing importance of electronic information management between businesses who use minicomputers.

If having access to information is the lock to a successul business, then teaming computer systems with autodialers is the key to that lock.

BIBLIOGRAPHY

[1]Peter B. Reintjes, "Network Tools, Ideas for Intelligent Network Software," BYTE magazine, vol. 6 no. 10 October, 1981, pg. 140

Programmatic Access to MPE's HELP Subsystem

John Cohen

HELPROC is the programmatic interface to MPE's HELP facility. It is callable from user mode programs. This ERS documents the usage of HELPROC for any users interested in adding a HELP facility to their application program.

HELPROC is not currently an intrinsic; it is a system-callable procedure. We will be considering adding it to the system intrinsic set in the near future. HELPROC has been in the system for many years; thus, we cannot consider changing its externals without considering backwards-compatability issues.

TABLE OF CONTENTS

ABSTRACT

MPE's HELP facility has a programmatic access that is callable from user mode programs. This access may be used with a previously prepared HELP catalog (such as CICAT.PUB.SYS) to provide a HELP facility for application programs. This document describes the usage of this system feature.

OVERVIEW

MPE's HELP command provides documentation on all MPE commands as well as some additional MPE features. A user may access this information in one of two ways:

1. The user may specify no parameters to the HELP command; in this case, the user will enter the HELP subsystem mode in which all command entries will be requests for additional information on the entered command (The user leaves the subsystem mode by entering "EXIT").
2. The user may specify one or two parameters to the HELP command; in this case, the user will receive the information requested and will be returned to the user's command interpreter for the next command entry.

The information displayed by the HELP command is stored in a catalog file called CICAT in PUB.SYS. This file is a regular, numbered EDITOR file that has been prepared as a HELP catalog by the HELP entry to MAKECAT.PUB.SYS. Unlike CATALOG.PUB.SYS, this file is not opened by the system; rather, it is opened only when needed (i.e., when a HELP command has been issued on the system). Thus, replacing the system's HELP catalog is done by purging the old CICAT.PUB.SYS, preparing a new catalog by running MAKECAT.PUB.SYS,HELP, and renaming the resultant file CICAT.PUB.SYS. The internal format of the HELP catalog and the usage of MAKECAT.PUB.SYS to prepare the catalog will be discussed later.

The Command Interpreter executes a HELP command by performing the following steps:

1. It opens CICAT.PUB.SYS,
2. It calls the procedure, HELPROC, passing it (among other things) the filenumber of CICAT, and
3. It closes CICAT.PUB.SYS when HELPROC is done.

HELPROC is a system procedure that is callable from user mode. It requires a filenumber of an opened, prepared catalog file as one of its parameters. Thus, a user wishing to add a HELP facility to an existing application program need only to do the following:

1. He must create a catalog file with the information to be displayed to the user,
2. He must prepare the catalog file with MAKECAT.PUB.SYS,HELP, and
3. He must modify the application program to do the following:
   a. It must open the prepared catalog,
   b. It must call HELPROC, and
   c. It may close the catalog when done.

The HELP facility in the application program will then behave in a manner similar to MPE's HELP facility. Thus, depending on the call, the application's HELP facility will either operate in "subsystem mode" or will return to the application program once specific information has been displayed.

The remainder of this document covers

1. The specifications of the procedure, HELPROC,
2. The internal format of a HELP catalog file, and
3. The methods used to prepare a HELP catalog with MAKECAT.PUB.SYS.

Note: some users may wish to merge information on various tools and applications with the system catalog, CICAT. These users may accomplish this by appending the desired information to the file and running MAKECAT again and replacing CICAT.PUB.SYS with the new, resulting file. In this way, the MPE HELP command can access information on that users' applications.

HELPROC Specifications

Accesses the MPE HELP facility.

Syntax

```
                IV        IV        BA        BA       I        LV
HELPROC( catnum, listnum, comimage, combase, err, onecharprmpt );
```

HELPROC accesses the HELP catalog specified by catnum and will display the information specified by comimage and combase to the file specified by listnum. Additionally, according to the information in comimage and combase, the HELP facility may be invoked in "subsystem mode" in which additional information may be requested until the user enters "EXIT" (which would then cause HELPROC to return to the calling procedure). This subsystem mode access is further specified in the Parameters section, below.

NOTE: The HELP catalog file must have been previously prepared by MAKECAT.PUB.SYS. The catalog file must be opened with foptions old, permanent, ASCII (foptions 5), and with aoptions nobuf and multi-record access (aoptions %420).

Functional Return: None.

Restrictions: No split stack calls.

Parameters

catnum — integer by value (required). A word identifier supplying the file number of the HELP catalog file. See NOTE above.

listnum — integer by value (required). File number (returned by FOPEN) of the file to which all HELP text is listed. Note that error messages and input prompts are always printed to the $STDLIST of the calling process. If listnum = 0, then all HELP text is printed to the calling process's $STDLIST.

comimage — byte array (required). This parameter points to a byte array character "string" specifying the information requested for this invokation of the HELP facility. This array must be terminated by a carriage return. If this string consists of any number of blanks followed by a carriage return, then the HELP facility enters "subsystem mode," in which the user will continuously request information until the HELP facility is terminated by the "EXIT" command. See description below in the Operation section for information on the expected format for information specification. Additional information may be found in the description of the internal catalog format.

combase — byte array (required). This parameter points to the "base" of the string used to invoke the HELP facility. Thus, if the user had entered "HELP SHOWOUT, ALL", this parameter would point to the "H" and comimage would point to the "S." This array is reused by subsystem mode.

err — integer (required). Word value to which an error number is returned if an error is detected by the HELP facility. These errors are those errors considered fatal to HELPROC; HELPROC can handle non-fatal errors like bad syntax in the input. HELPROC prints its own error messages for all errors detected. These fatal errors are

42→End-of-file on $STDIN.
51→Error detected in reading catalog file.
52→No user labels for directory in catalog. (Catalog file may not have been prepared by MAKECAT.PUB.SYS).
53→No directory in catalog file user labels. (Catalog file may not have been prepared by MAKECAT.PUB.SYS).
54→Bad format of directory in catalog file user labels or error detected in attempting to read directory. (Catalog file may not have been prepared by MAKECAT.PUB.SYS).
55→Unable to open $STDIN.
56→Error detected in attempting to read from $STDIN.
57→Error detected in attempting to write to the HELP text list file.

If no fatal errors are detected by HELPROC, a zero is returned to err.

onecharprmpt — logical by value (required). Logical word value that specifies that a one character prompt was used with the command invokation of the Help facility. If a one character prompt was in fact used, then any error messages may have a caret ("^") printed under the part of the invokation in error. If some other number of characters was used for the prompt, then the Help facility will not be able to place the caret in the correct location — thus, if FALSE is the value specified for this parameter, no caret is printed with any error message.

Condition Codes

Not returned by this procedure.

Operation

This section discusses the operation of the HELPROC procedure; refer to later sections for information on how to format a catalog file and how to prepare the file for use.

Information in a help catalog is organized in an heirarchical structure. The help catalog consists of a set of entries; each entry has optional header information and an optional number of items; each item has optional header information and an optional number of subitems. Additionally, each catalog has a special table of contents entry (typically called "HELPMENU") — this entry is printed by default when the HELP facility is entered in subsystem mode. This entry is always the first entry in the catalog; its item and subitem names are considered part of the table of contents.

Invokation of the HELP facility may take many forms. These are specified by the contents of the parameter, comimage. The varieties of the different calls and a brief description of each are presented below. Note that comimage is always terminated by a carriage return.

Comimage contents **Description**

(blank line) — Print table of contents entry header, and enter subsystem mode. Keyword list consists of all the item and subitem names of the table of contents entry.

Table of Contents—Item Name — Print all the information in that item's header and all the information in all the associated subitems. Keyword list consists of all the item and subitem names of the table of contents entry. HELPROC terminates when printing is done.

Table of Contents—Subitem Name — Print all the information in that subitem's block. Keyword list consists of all the item and subitem names for the table of contents entry. HELPROC terminates when printing is done.

"ALL" — Print all the information in the table of contents entry including all item and subitem information. Keyword list consists of all item and subitem names of the table of contents entry. HELPROC terminates when printing is done.

Entry Name — Prints all the information in the header for the associated entry. Keyword list consists of all the item and subitem names for that entry. HELPROC terminates when printing is done.

Entry Name, Item Name — Prints all the information in the indicated item's header as well as any information in that item's subitems. Keyword list consists of all the item and subitem names for that entry. HELPROC terminates when printing is done.

Entry Name, Subitem Name — Prints all the information in the indicated subitem. Keyword list consists of all the item and subitem names for that entry. HELPROC terminates when printing is done.

Entry Name, ALL — Prints all the information in the indicated entry including all item and subitem information. Keyword list consists of all the item and subitem names for that entry. HELPROC terminates when printing is done.

In subsystem mode, the HELP facility performs the same as above with two exceptions: (1) HELPROC does not terminate after printing information — the user must enter "EXIT" to terminate the procedure; and (2) a blank line will cause the next block of information to be printed — thus, when the user enters an entry name, that entry's header information is printed; if the user then enters a blank line, that entry's first item information will be printed (after the last item has been printed, a blank line will cause the next entry header to be printed). Any fatal error will terminate subsystem mode; note that being unable to find the requested information is not considered a fatal error.

In all cases, HELPROC will open $STDIN, but $STDIN is only used in subsystem mode — HELPROC reads user requests for information from $STDIN into combase.

Helpful Hints

1. Testing your catalog file: You need not have your program ready in order to determine whether your catalog is formatted correctly. Format your file with MAKECAT.PUB.SYS (documented in a later section). If MAKECAT determines that your catalog file is syntaxically correct, you may use MPE's HELP command to view it. Suppose the resultant catalog file from MAKECAT is called "MYHELP." Enter the following :FILE command:

```
:FILE CICAT.PUB.SYS = MYHELP
```

   Then, for the rest of your session (or until you :RESET the :FILE equation), HELP commands will open your catalog instead of MPE's HELP catalog. It is often useful to view your catalog file in this manner after running MAKECAT.PUB.SYS. In order to restore MPE's HELP command back to its original condition, enter the command

```
:RESET CICAT.PUB.SYS
```

   Note that this method of testing only effects your session. Other jobs and sessions will be able to access CICAT.PUB.SYS while you are testing your catalog.

2. Testing your program: Similarly, you need not have your catalog ready in order to test the HELP facility in your program. Assuming the same

names in the example above, you may issue the following :FILE command

```
:FILE MYHELP = CICAT.PUB.SYS
```

Since you FOPEN of the catalog file in your program should allow :FILE equations, this command would open MPE's HELP catalog instead. This is often useful in testing your program.

Stack Size: The HELP facility uses a directory to find information in the catalog file. This directory must fit on the process's stack. The space needed by the HELP facility currently is 3650 words. Use of the intrinsic, ZSIZE, before a call to HELPROC can assure that there is sufficient stack space for HELPROC.

INTERNAL CATALOG FORMAT

Overview: The source file for a catalog file should be either another catalog file or a standard, numbered EDITOR file. MAKECAT creates the resulting file as a new file; thus, the group which holds the resulting catalog file should have ample file space. Further, MAKECAT inserts new records into the file in certain circumstances (see CONTINUE, below). Thus, it is advisable to have ample line number space between the lines of the source EDITOR file.

All record lines that have a backslash ("\") in column one are considered to be an internal directive to MAKECAT and HELPROC. Thus, no text lines can have a backslash in column one. The backslash is followed immediately by a keyword and some other parameters. The current keyword list and a brief description is given below; a more detailed description for each follows.

```
Keyword         Description
\ENTRY          Start of the next entry text block.
\ITEM           Start of the next item text block.
\SUBITEM        Start of the next subitem text block.
\STOPHELP       Initiates skipping of text in building catalog.
\STARTHELP      Undoes the effect of previous \STOPHELP.
\SUBSET         Causes text between \STOPHELP and \STARTHELP to
                   be removed from resulting catalog.
\CONTINUE       Provided by MAKECAT to handle overflowing \ENTRY
                   lines.
\ALL            Indicates the end of the catalog file.
```

MAKECAT modifies these records freely. The syntax reported below is the syntax expected in the source catalog. The discussion for each keyword explains the modifications made by MAKECAT. In all cases, MAKECAT can take a prepared catalog as its source catalog — MAKECAT ignores its own modification present in the source catalog.

\ENTRY

```
Syntax:  \ENTRY=<entryname> [, <other information> ]
```

The presence of this record terminates the text to the previous entry; the next non-keyword record (a line not starting with a "\") is the first line of the entry text for an entry called "<entryname>." other information present on this record in the source catalog file will be ignored.

MAKECAT echoes this line in the resulting catalog file with some modifications. The keyword list (i.e. all item and subitem names) for this entry is appended to the end of the line. If the keyword list overflows the line, MAKECAT will create a CONTINUE line to hold additional keywords. If sufficient line number space is not available following the ENTRY record for any needed CONTINUE record, MAKECAT will not be able to format the catalog; an error message is issued in this case.

\ITEM

```
Syntax:  \ITEM=<itemname>
```

If this keyword record is the first keyword record following a ENTRY record (exception: see STARTHELP and STOPHELP), then this record terminates the "header" text for that entry; otherwise this record terminates the text for the previous item or subitem. The following non-keyword records will comprise the text for the item called "<itemname>." This text, in turn, will be terminated by either the next ITEM record, the next ENTRY record, or the end of the file, whichever comes first.

"<itemname>" becomes a keyword for the previous entry and gets added to the ENTRY record or the associated CONTINUE record.

\SUBITEM

```
Syntax:  \SUBITEM=<subitemname>
```

If this keyword record is the first keyword record following a ITEM record (exception: see STARTHELP and STOPHELP), then this record terminates

the "header" text for that item; otherwise this record terminates the text for the previous subitem. The following non-keyword records will comprise the text for the subitem called "<subitemname This text is considered part of the text for the previous ITEM record, and is, in turn, terminated by the next SUBITEM record, the next ITEM record, the next ENTRY record, or the end of the file, whichever comes first.

"<subitemname>" becomes a keyword for the previous entry and gets added to the ENTRY record or the associated CONTINUE record.

\CONTINUE

Syntax: \CONTINUE,<keywordlist>

The presence of a CONTINUE record in the source catalog is ignored by MAKECAT. These records are created when the keyword list for a ENTRY overflows the ENTRY record. MAKECAT attempts to place the CONTINUE record directly after the ENTRY record, or if other CONTINUE records have already been created, after the last CONTINUE record. If line number space does not exist for the creation of the CONTINUE record, MAKECAT prints an error message and terminates.

\STOPHELP

Syntax: \STOPHELP

The presence of a STOPHELP record will cause MAKECAT to ignore records in the source catalog until it finds the next STARTHELP record. These records will be copied to the resulting file (exception: see SUBSET), but the text and the keywords are ignored by both MAKECAT and HELPROC. This feature is used to void blocks of text from the source catalog.

\STARTHELP

Syntax: \STARTHELP

The presence of a STARTHELP record will cause MAKECAT and HELPROC to resume processing the text and keyword records in the source catalog. Note that a STARTHELP record need not follow a STOPHELP; thus, a single STARTHELP can be used to void a single line in the source file.

\SUBSET

Syntax: \SUBSET

The presence of a SUBSET record will stop the copying of text between a STOPHELP and a STARTHELP (as well as all STOPHELP and STARTHELP records themselves) from the source file to the resulting file. This is used to compress the file size.

\ALL

Syntax: \ALL

This record terminates the source catalog. It must be the last record in the source file.

EXAMPLE SOURCE CATALOG

The following is an example of a small source catalog. The line numbers are added for reference and may be assumed to be EDITOR line numbers.

```
 1  \entry=helpmenu
 2      This is the text for the "header" of the
 3  catalog.  This text will be printed when the
 4  HELP facility is entered in subsystem mode.
 5  \item=jobs
 6      This is the text for the "header" of an
 7  item called "jobs".
 8  \subitem=limit
 9      Subitem "limit" text.
10  \subitem=logon
11      Subitem "logon" text.
12  \item=sessions
13      This is the text for the next item, which
14  is called, "sessions".
15  \stophelp
16      All text here is ignored and is unaccessable
17  from the HELP facility.
18  \starthelp
19      This text is the continuation of text for the
20  item call "sessions".
21
22  \entry=usage
23      This is a new entry heading.
24
25      The next \line will terminate the help catalog,
26      these lines are text for "usage".
27  \all
```

The appearance of the resulting file once MAKECAT.PUB.SYS in run will be essentially the same as above with the exception of line 1. This line will appear in the following form:

```
1 \entry=helpmenu,jobs,limit,logon,sessions
```

Had a " subset" line been added at line number 0.1 (and if the SUBSET facility was working properly), the resulting file would have had lines 0.1, and lines 15 through 18 deleted.

PREPARING THE HELP CATALOG WITH MAKECAT.PUB.SYS

The system utility program, MAKECAT, (typically residing in PUB.SYS) is used to prepare a catalog for use by HELPROC. Unless you are replacing the system HELP catalog, CICAT.PUB.SYS, no special capabilities are needed to use MAKECAT. The "HELP" entry point must be used.

Preparation of the catalog consists of syntax checking of the catalog format and the creation of a entry directory to be stored in the user labels of the resulting catalog file.

MAKECAT uses two formal file equations in preparing help catalogs. The file called INPUT is the source catalog, and the file called HELPCAT is the resulting catalog that may be used with HELPROC. Thus, if a user has a file called SCAT1 that he/she wishes to use as a help catalog and if the user wishes the resulting help catalog to be called HELPTEXT, then the following command sequence prepares the catalog:

```
:FILE INPUT = SCAT1
:FILE HELPCAT = HELPTEXT
:RUN MAKECAT.PUB.SYS,HELP
```

If MAKECAT produces the message, "VALID MESSAGE FILE," then the file called HELPTEXT (produced by MAKECAT) can be used with HELPROC. If MAKECAT is unable to produce a valid catalog file, it will produce an error message and terminate in the error state. The error conditions reported by MAKECAT for HELP catalog preparation include

```
** FILE ERROR ON INPUT:  Unable to either open or access the file
            associated with the formal designator, INPUT.

** FILE ERROR ON CATALOG:  Unable to open, close, or access the file
            associated with the formal designator, HELPCAT.

** FILE ERROR ON LIST:  MAKECAT opens a file called LIST (default
            to $STDLIST unless overridden by a :FILE equation).  This
            error is reported if MAKECAT cannot open the file.

** CONTINUATION FREADDIR ERROR   and
** CONTINUATION FWRITEDIR ERROR:  MAKECAT needs to shuffle around the
            directory in order to make room for \CONTINUE records.
            This error is reported in MAKECAT has trouble accessing
            the directory in the file's user labels.

** INSUFF. LINE # SPACE FOR CONTINUATION:  MAKECAT creates extra
            lines for \CONTINUE records after \ENTRY (or other
            \CONTINUE) records by incrementing the eight character
            sequence number in the last eight positions of the
            record line.  This sequence error occurs when the next
            record's sequence number does not allow the creation
            of the new record.  The solution in this case to to
            text the file into the EDITOR and renumber the file with
            a "GATHER ALL" command.  (Note:  EDITOR line numbers when
            viewed as the eight character sequence number at the end
            of record are incremented by 1000 between integer line
            numbers--this is more than ample space for the \CONTINUE
            records which would be incremented by .001 as the EDITOR
            does line numbering.)

** KEYWORD LIST WON'T FIT:  MAKECAT detected a keyword too large to
            fit onto the \ENTRY or \CONTINUE record.

** OVERFLOWS DIRECTORY.  REC= :  MAKECAT produces a directory for HELP
            catalogs to be placed in the user labels of the resulting
```

file. The original HELP catalog file is created with a default number of user labels for the directory--if more user labels are needed, a new catalog file is created. This error is reported if the directory cannot fit into the maximum number of user labels per file (255).

** UNABLE TO OPEN LARGER FILE
** UNABLE TO READ DIRECTORY
** ENLARGED DIRECTORY TOO SMALL
** COPYING ERROR (DIRECTORY)
** UNABLE TO 'REWIND' OLD FILE
** READING ERROR WHILE ENLARGING
** ENLARGED FILE TOO SMALL
** COPYING ERROR (FILE): As mentioned above MAKECAT has to open a new file when it needs to enlarge the number of user labels for the directory. When this new file is opened, the information in the old file, both directory (user labels) and records, has to be copied to the new file. The old file is then deleted. These errors may be reported for situations in this operation.

** MISSING '\ALL' AT THE END OF HELPSET: The \ALL record was missing from the end of the source file.

Refer to the System Manager/System Supervisor Manual for more information on running MAKECAT.

KNOWN ERRORS IN THE HELP FACILITY

1. The SUBSET facility does not work — the presence of a SUBSET will render the rest of the catalog following the next STOPHELP useless.
2. When MAKECAT cannot format a catalog for any reason, it should terminate in the error state (thereby setting the system-defined JCW to a value greater than FATAL).
3. As of this writing, HELPROC is not an intrinsic, but it is a system procedure that can be called from user mode. In order to call it, HELPROC needs to be declared as an EXTERNAL procedure. Here is an example of how to declare HELPROC:

```
PROCEDURE HELPROC( CFN, LFN, CIM, CBASE, ERR, PRMPT );
   VALUE   CFN, LFN, PRMPT;
   INTEGER CFN, LFN, ERR;
   BYTE ARRAY CIM, CBASE;
   LOGICAL PRMPT;
OPTION EXTERNAL;
```

4. The error reporting in MAKECAT should be improved.
5. HELPROC need falling-off-the-end-of-the-world checking to insure a carriage return in the byte array.
6. HELPROC should check that catalog was opened correctly.
7. HELPROC should only open $STDIN in subsystem mode.
8. HELPROC makes no check for split stack calls.
9. Listfile parameter is always ignored. All output goes to the caller's $STDLIST.

SUMMARY AND CONCLUSIONS

The user may use HELPROC with a prepared catalog as a help facility for application programs. Catalogs are prepared with the HELP entry point of MAKECAT.PUB.SYS. In order to use this help facility, the user should

1. Create a source text catalog file in the documented format,
2. Prepare the catalog by running MAKECAT.-PUB.SYS,HELP and,
3. Modify the application program to
   a. Open the prepared catalog file,
   b. Call HELPROC,
   c. Optionally close the catalog file when done.

The Help Facility is designed to general and flexible. Use of MPE's Help Facility can be extended for application programs.

Management Options For The 80's

Giles Rider
Project Manager Rex Development

Mr. Rider has been reponsible for development of PAL — the User-friendly Front-end to REX/3000, Gentry, Inc.'s general purpose programming language and report writer. Gentry, Inc. is a HP OEM and Contracting company in Oakland, Ca. Mr. Rider was educated at MIT and the University of Connecticut. Prior to joining Gentry, Mr. Rider was an independent consultant.

INTRODUCTION

HP's choice of the phrase "Interactive Information Management" as it's theme for the beginning of the 1980's reflects the changing emphasis and the quickening pace of the industry as a whole. By the middle of the decade much of today's hardware and software will be obsolete, and many traditional DP concepts will also have fallen by the wayside. The continuing rush of hardware developments — increasing speed and reducing cost per calculation — make it practical to cost-justify new applications daily. New powerful software — Data Dictionaries, report Generators, turnkey applications — holds out a lot of promise for simplifying application development. Yet each new application presents risks - both personal ones, in terms of job security, and company risks, in terms of survival and growth of the business.

In this paper we seek to identify guidelines and principles that can be followed to minimize these risks — guidelines that will also help to identify new opportunities as they arise, and to exploit them fully. We will examine each of the areas that are mentioned in the title and attempt to provide ideas for action that are both practical and useful. These ideas are intended to supplement, rather than to supplant the conventional wisdom on each of the following subjects, while providing a unified viewpoint — a grand scheme — from which it all makes sense.

Before we begin the detailed analysis of the areas, a few general remarks are in order. Economics indicates that the scarcity of a resource determines its price, and that cheaper resources are always substituted for more expensive ones when they are available. When the CPU was the most expensive component of a system and software was cheap, it made sense to use keypunches and run all jobs in batch mode; now it doesn't. Now that maintenance of old applications is the single largest people cost in DP, does it still make sense to keep building applications that will need to be maintained in the traditional ways? We saw that CPU power was substituted for people power in the transition from batch to on-line; doesn't it make sense economically now to begin to substitute CPU power for people power in the area of system maintenance?

Applications development is another area where people cost is the major expense — and this isn't because application development is the kind of "creative" activity where only "artists" can do the work. More often than not, the main reason for the cost is that suitable automated tools either do not exist or are not used, so that manual reports are the mainstay of most development projects. Many automated project control systems stress accounting for the cost of the project more than supporting the design and implementation activities, which reflects DP's traditional accounting-support function and orientation.

The use of Data Dictionaries in applications development represents another application of CPU power to reduce the effort involved in application development — and in maintenance as well. In application development, the use of dictionaries tends to bring disagreements about the use and meaning of the data out early enough in the project so that the conflicts can be resolved in the system design, rather than by after-the-fact administrative procedures and manual forms. In the maintenance area, where about 70% of the programming dollars are currently spent, a properly constructed dictionary can result in enormous savings, as the dictionary is the only place changes to applications have to be made by hand, and proper safeguards can be set up and followed to ensure that the integrity of the databases is not compromised.

Application packages can also help to ensure that the limited DP budget buys the most in DP service. Application packages cost so much less than "home-made" programs that the justification of "in-house" programming for common applications like A/R is very difficult, especially for new users. Another powerful reason for moving to packages is that the shortage of programmers makes it disadvantageous to waste them on re-inventing a General Ledger system, when they could be creating a new system for your company that could give it a lead on its competition. Programmers know this too, and are more likely to stay with companies that promise them a future working on new development and "state-of-the-art" projects than with companies that offer maintenance of an obsolete G/L system.

Contract programmers are about the only ones who are willing to get involved with maintenance programming as a steady diet, and if you use them properly, they can benefit your company, allowing you to make more profitable use of your regular employees, while not causing salary riots because of their higher rates, because they are doing the work that no one else wants to do.

Microcomputers in user departments are often seen as a symptom of DP department unresponsiveness — a "sagebrush rebellion" in the user area. There's an old saying that a problem is just the wrong way of looking at an opportunity, and that's especially true of the Microcomputer. A proper response to the Micro invasion can actually improve the situation, both for the DP department, and for the users; cutting costs while building user confidence in your DP expertise.

Programmer productivity tools, software tools, Programmer's workbench — all of these buzzwords carry thru the concerns that we've previously mentioned — how to apply CPU power to make the programmer's job easier. But there's two sides to this one, and there's good reason for caution. Here's where the approach that de-skills the programming function can cause the same kind of worker problems that plague the assembly line factory and the so-called "office of the future." Yet good productivity tools do exist, and should be used whenever possible to reduce the drudgery and repetition that is often found in programming. Programmers are one of your most expensive resources, and to watch one using a line editor can be a real eye-opener.

DATA DICTIONARIES:

Rule #0: Select the right data dictionary for your environment.

Data Dictionaries differ in capabilities and in their interfaces to the outside world. Ideally, you want a Dictionary that allows you to create your application code via the most powerful application generator that your environment will allow (this could be a manual application generator — i.e. a programmer — or an automated one that produces COBOL or other source code.)

Your dictionary will also be a central place for manual changes — when the length of a data item changes, for instance, you should only have to change the entry in the data dictionary, not in the programs that access the data. You'll see the importance of this when you remember that in many shops these kinds of maintenance costs make up 70% of the programming bills.

Rule #1: Get a technically strong and personally forceful person in charge of the Data Dictionary.

The "Data Administrator" is the person who is going to have to tell a lot of people in your company that they can't have what they want, and that their use of a data element conflicts with another use of the element. A person who is too weak to do this can result in your dictionaries and applications becoming complex very quickly. He's also going to have to deal with sneaky programmers who will try to use data elements in strange and undocumented ways; he has to be forceful enough to show them that they can't get away with it.

Rule #2: Use the data Dictionary at the beginning of the development Phase to detect and resolve conflicts in requirements.

If each data element is added to the dictionary as it is defined you can use the dictionary's reports to detect conflicts and in-consistencies. Here's where your strong "data administrator" gets called in to resolve the conflicts, before your programmers go ahead and build the conflicts into the application, or devise some arcane method to get around the conflicts that will work until they leave.

Rule #3: Create and enforce standards and procedures for the Dictionary.

Your staff won't give the dictionary any more respect than you give it; if you make it important and vital, they will follow you. Dictionary updates should be in the hands of the data administrator only; this will give him enough power to deal with the programmers and analysts. Standards for development that ensure that no-one can by-pass the dictionary are also vital to proper maintenance of the developed systems. Ideally, nothing should go into production that isn't fully described in the dictionary.

Rule #4: Retrofit existing systems into the dictionary in order of their estimated lifetime.

"Doomed" systems should go in first, if they are to be replaced. Stable, low maintenance systems have lowest priority, while systems due for extensive maintenance should be dictionarized before the maintenance is started (see rules 2-3 for why this is a good idea.)

APPLICATION PACKAGES:

Rule #5: Let your users select the package they want, from the best ones that you know are available.

With any new application, there will be problems. If your staff is programming the application for the user, the problems will be your problems, affecting your budget and your schedule. If you can avoid being the cause of problems, you'll be able to continue to be of service to your company.

Rule #6: Select a strong member of your group to serve as liason to the user group.

Ideally, this will be someone strong enough to be promoted out of your group and into user management, where he can carry your message into areas that you can't reach, and help build support for you in the various user departments. Remember, your representative speaks with your voice, so don't send a pure technician; send a heavyweight.

Rule #7: Ensure that the users contract with the package vendor for maintenance.

This is important enough to justify rule #6, for users are never happy with the DP's response to maintenance requests. They hate all the paperwork involved. Managing maintenance is a classic "no win" situation — no matter how it's handled, users are never satisfied, so whenever possible, let the vendor handle it. The first task of your "strong" liason person is to convince the user to go with a vendor maintenance contract; let him know that if he fails, he's going to be doing the maintenance for the user.

MICROCOMPUTERS IN END-USER DEPARTMENTS

Many DP shops see Micros as a real menace to the DP department's control of the corporate information resource. They try to make it difficult or impossible for end-users to get and use Micros by blocking any purchases or evaluations of them.

This "dog-in-the-manger" approach to the situtation will be self-defeating in the long run, with serious consequences to the DP department's credibility and stature within the organization. It's far better to encourage end-users to use Micros properly, and to support them in their efforts. You can build your credibility within the organization, while improving the profitablity of the the company by supporting the judicious use of the proper mixture of computer resorces.

Rule #8: Don't fight user-owned Micros — encourage them, instead.

Development of a system is a small part of the total effort involved; maintenance and operations are far bigger pieces of the job. What happens in the long run is that the systems that the users develop on their micros become prototypes for systems that can be developed on the mainframes, if the users have developed a succesful system. If their development effort was a failure, it is buried within the user department; the only systems that the DP department will have to support will be the succesful ones that have outgrown their micro, and need a larger machine to work on.

Rule #9: Encourage your technical people to get involved with micros. This is one rule that it probably will be easy to enforce. Let your programmers help out the users with their technical questions about their micros; encourage your technical people to become familiar with the succesful applications, so that when the time comes to put that application up on the mainframe, they'll know how to do it.

Rule #10: Use micros yourself if you haven't already.

Try out the HP-125; the VISICALC package can make your budgeting easier. You will also get a better understanding of why your users want to by-pass the whole DP department applicatio development process with its paperwork and problems.

APPLICATION GENERATORS:

You already have some manual ones in your shop — the programmers. They can be used to create both simple, repetitive reports or complex applications systems, but they can't do both at the same time. They can also be your liason with the user departments, or write simple, repetitive reports and database update programs. An application generator can free them up to do the things that are more important for the long-range success of the department and the company. It can also shift some of the report-creation into the user departments and out of your budget.

Rule #11: Identify and forecast your programming needs.

If a substantial part of your work is either

A. Ad-hoc report requests or
B. Moderately complicated systems,

then an application generator can ease your burden and reduce your costs. (If your applications needs are simple, you probably should be looking more towards packaged software than to applications generators.)

Rule #12: Select a generator that can be used by both programmers and end-users.

End-users who can produce their own reports using applications generators and who can do their own data entry, are getting what they need from the computer without involving the DP department. Their applications are part of their budget, not yours. Programmers who can use an application generator will typically use it like a ladder, to extend their reach. An application generator that can handle most of the boring detail work reduces the number of programming errors dramatically because boredom causes errors. Application generators can also be very useful with trainee programmers, to take them thru the process of writing a program from specifications by examples.

Rule #13: Select an application generator that is compatible with your Data dictionary.

The application generator should make the maximum practical use of the data dictionary to reduce the drudgery of programming and to control the access to the data in some standardized way. Ad-hoc report generators that interface to the dictionary must provide for separate logical views for each user, as most users have different reporting needs, and do not need extraneous detail.

USING CONTRACT PROGRAMMERS

Many shops are reluctant to use contract programmers but the proper use of contract people can multiply your productivity. One criticism of contract people is that the expertise goes away with them; this criticism conceals an error about what is important. The expertise that is important to the business is the knowledge of the applications, not the knowledge of the computer and

its processes. At any time, you could switch computers, but the business will continue to do what it has been successful at.

This fundamental truth underlays many of the rules that I have suggested so far — that you try to build a staff of people that know the business. This is why contract people can be useful, because you can bring them in to do work that is only technical — language conversions, maintenance, etc, while your stronger staff people work with your users to develop an understanding for and appreciation of the business.

Rule #11: Hire the company, not the programmer.

Find a contracting company that understands your business and your needs and build a long-term relationship with them. You'll soon find that a seasoned professional contractor costs less in the long run than a trainee employee, unless your plan for the trainee is user-support work. Hiring trainees to do maintenance is a sure plan for building plenty of fires to fight, and lots of turnover in the long run.

Rule #12: Use contractors as followers, not leaders.

The best use for contract people is on temporary jobs that have a clearly defined beginning and end; in development projects, they should code programs, set up databases, do documentation, or even be project leaders, but they should not be in charge of defining the user's requirements, as you want to keep the business expertise in-house.

It's also good to put contract people on maintenance work, to reduce your employee turn-over, and the cost should always be billed back to the user department budget. This can be a positive morale-booster for your regular people, who will be getting only the interesting work.If one of your regular employees wants to leave you to become a contractor, with the hope of making big money doing maintenance work, then you're probably better off, because that's not the kind of employee that you should be trying to develop.

PRODUCTIVITY TOOLS

A software tool is a computer program that a programmer uses to do something useful for someone. The first programmer tools were things like COBOL and SORT programs, while current programmer tools include preprocessors that generate programs from pseudocode. The basic thrust is to automate some simple, but repetititous part of the programmer's job, as opposed to the alternative approach of breaking the job down into simple parts and using an assembly-line technique to do the work.

Programmers like software tools because their jobs become less boring; most people are drawn to programming because of the creative aspects of the job, and a properly crafted tool multiplies their power to create. In addition, tools can substitute for other changes in working conditions by providing a work atmosphere that is more pleasant, in the sense of less boring work.

Rule #13: Start with the Data Dictionary.

The role of the Data Dictionary is so central to system development that it is the one productivity tool that you shouldn't try to do without. (see rules #0-#4 for clarification) A properly used dictionary can create savings far in excess of its cost, while reducing development time proportionately. The Dictionary can be so powerful a tool, that the question becomes not "Should I get a Dictionary" but "Which dictionary should I get"?

Rule #14: Get a preprocessor that inerfaces to the Dictionary.

The combination of a dictionary and an preprocessor can reduce your programming effort by 30-70% almost immediately, allowing you to get rid of your backlog of user requests quickly, without hiring additional staff. One very important point is a compiler-data dictionary interface, so that the physical location and characteristics of each data item are taken from the data dictionary at compile-time — which effectively eliminates most maintenance programming.

Rule #15: Enforce structured programming, via a preprocessor or use of a PASCAL-like language.

Structured programming, in the narrowest sense of GOTO-less coding, results in easier-to-maintain code and easier-to-debug applications. Programmers, however, are lazy and will use a GOTO anytime they think themselves into a corner in a program and can't find a simple logical way out. That's one big reason why GOTO-less coding is easier to maintain — programmers don't get to take the easy way out in the first place.

PASCAL and it's relatives are what is being taught in school these days, so using a PASCAL-like language offers the added benefit of not having to train programmers in COBOL, while offering a "state-of-the-art" programming shop as a morale-builder and PASCAL offers the GOTO-less syntax that enforces structured programming painlessly.

CONCLUSION

The set of rules given above is based on the idea that your people, and their knowledge of your company's business, constitute your most important resource. The technical aspects of computing are constantly changing, while the business changes much more slowly. The secret to success is to offer your people an environment where their professional growth is directly related to the growth of the business, and to realize that you can't be promoted until you've found someone who can take your place.

Transaction Processor For The HP3000

Godfrey lee
Project Manager
Quasar Systems Ltd.

I am sure that each of us has had the need to manipulate files, or perform bulk updates of an application database, and found that the existing methods are either incomplete (i.e. FCOPY) or too troublesome (i.e. COBOL) to use. Most application systems involve several standard batch functions which require custom programming. Yet the task involved is so standard one should be able to specify it in a simple, logical and straightforward manner.

These functions can include:

- Daily, weekly, or monthly rollovers,
- Reformatting a file,
- Producing a summary file,
- Selectively copying based on some condition,
- Copying elements from one file to another,
- Reformatting a database.

File manipulation tasks are a common requirement in developing as well as running most application systems. While excellent productivity tools now exist for large segments of application development and maintenance, batch processing programs still have to be prepared in the same tiresome manner.

The paper introduces the concept of a powerful batch-oriented data manipulation tool called a TRANSACTION PROCESSOR, which will keep pace with and interface with current state-of-the-art productivity tools.

The TRANSACTION PROCESSOR will be called QTP and will complete Quasar Systems' family of application generator products, which currently include QUIZ for reports and QUICK for screen-based input. With this family, users will be able to generate entire applications in a consistent easy-to-use style.

This paper will discuss:

1. QTP in relation to an application dictionary
2. Design objectives
3. QTP in operation (some examples)
4. QTP in the production environment
5. Design considerations.

TRANSACTION PROCESSOR AND THE APPLICATION DICTIONARY

The transaction processor will be able to operate as an independent product in association with Quasar Systems' application dictionary. In essence, QTP will have two components:

1. QSCHEMA

The schema processor compiles a description of data files and element characteristics including data validation and display specifications. The compiled schema functions as an application dictionary, providing central administrative control and freeing users of QTP from a great deal of repetitive programming.

2. QTP

Under the control of specification statements which can be used by both programmers and non-programmers, QTP will carry out two major functions:

- standard batch applications
- file manipulation.

DESIGN OBJECTIVES

The design objectives of QTP are:

- to support the standard maintenance functions of add, change and delete against all data permanently on file
- to support standard editing of input including, type checking, value range checking, and pattern matching
- to support the copying of elements from one file to another
- to allow the reformatting of files and databases
- to be able to produce summary files
- to support these summary options: sum, count, average, maximum, minimum, percentage and ratio
- to be able to specify the sorting and selection of input files
- to support any combination of IMAGE, KSAM and MPE files
- to reference the structure, composition and elements of files in a central independent schema
- to use concise specification statements in simple free-form syntax.

The Transaction Processor in Operation

To show the scope of the QTP in operation, here are four short examples of situations which occur frequently and which normally require specially written programs.

1. New product number

The manager of inventory control wants to assign a new product series "M" to all series "S" product numbers greater than 6000. With QTP, this task could be accomplished by entering the following statements:

```
>ACCESS PRODUCTS
>SELECT IF PRODUCT-CODE = "S" AND PRODUCT-NUM > 6000
>FILE PRODUCTS UPDATE
>  ITEM PRODUCT-CODE FINAL "M"
>GO
```

The ACCESS statement specifies which file(s) are to be read-in this case, the file PRODUCTS. The SELECT statement then restricts the selection of records from the product file to those records to be changed. The FILE and ITEM statements specify the changes to be made to selected records. The GO statement causes the QTP request to be executed.

2. Organizational change

The San Francisco branch has been reorganized and is now part of California branch. All reference to San Francisco is to be deleted and all records for San Francisco employees are to be updated to reflect their new status as records of California branch employees.

```
>ACCESS BRANCHES LINK TO EMPLOYEES LINK TO BILLINGS
>SELECT IF BRANCH-NO OF BRANCHES = "SF"
>FILE EMPLOYEES UPDATE
>  ITEM BRANCH-NO FINAL "CA"
>FILE BILLINGS UPDATE
>  ITEM BRANCH-NO FINAL "CA"
>GO
```

The ACCESS statement in this example illustrates multi-file access. Keyed linkages between files can typically be performed automatically, using information in QSCHEMA. The ITEM statements in this example set BRANCH-NO to "CA" in the selected EMPLOYEES and BILLINGS records.

3. Culling obsolete data

A company wants to streamline their customer file and delete anyone on their mailing list who hasn't corresponded for over a year.

```
>ACCESS MAIL-LIST
>SELECT IF 365 < (DAYS (SYSDATE) - DAYS (RESPONSE-DATE))
>FILE MAIL-LIST DELETE
>GO
```

The FILE statement in this example deletes all records of MAIL-LIST that have satisfied the condition in the SELECT statement.

4. Reformatting a file

QTP will be ideally suited to problems involving the reformatting of files. Assume for instance, that the old customer file shown in Figure 1 is obsolete. The "PYR-SALES" (previous years sales) item is to be dropped; "YTD-SALES" (year to date totals) is to be expanded for larger dollar volumes; item "CUSTOMER-ID" is to be expanded; an item "SALESMAN-CODE" is to be added; and all items are to be re-ordered.

```
OLD CUSTOMER MASTER                NEW CUSTOMER MASTER

CUSTOMER-NAME        X(20)         CUSTOMER-ID          X(10)
CUSTOMER-ID          X(6)          CUSTOMER-NAME        X(20)
CUSTOMER-ADDRESS     X(60)         CUSTOMER-ADDRESS     X(60)
PYR-SALES            9(6)          SALESMAN-CODE        X(6)
YTD-SALES            9(6)          YTD-SALES            9(10) COMP
```

Figure 1

The steps needed to format the new customer file are:

(a) Unload the master file.

```
>ACCESS CUSTOMER
>SUBFILE TMP OUTPUT CUSTOMER
>GO
```

(b) Change the schema, purge and recreate the customer file (details not shown).

(c) Reload the new master file.

```
>ACCESS *TMP
>FILE CUSTOMER ADD
>GO
```

(d) Purge the temporary file.

```
:PURGE TMP
```

The SUBFILE statement creates an ad-hoc file containing specified information. Subfiles automatically contain their own schema and are therefore self-describing. In this example SUBFILE creates a temporary file TMP containing a copy of the customer master file.

QTP automtically performs the following manipulations for commonly named items in the two files:

- changes item type
- changes item size
- changes item order.

QTP IN THE PRODUCTION ENVIRONMENT

The following two examples look in detail at how QTP might handle two common month-end production situations.

1. Adding 1% interest to all invoices over 30 days due

To expand on a typical accounts receivable situation, assume a company has reached the due date for monthly accounts receivable. The account manager wants to add 1% interest to all outstanding accounts and update the master file. QTP performs this task in fewer than 20 specification lines.

```
>ACCESS ACCOUNT-MASTER LINK TO ACCOUNT-DETAIL
>SORT ON ACCOUNT-NO, INVOICE-NO
>TEMPORARY INVOICE-DATE RESET AT INVOICE-NO &
>  INITIAL DATE OF ACCOUNT-DETAIL &
>                IF TYPE OF ACCOUNT-DETAIL="INVOICE"
>TEMPORARY INVOICE-BALANCE RESET AT INVOICE-NO INITIAL 0
>SUM AMOUNT OF ACCOUNT-DETAIL INTO INVOICE-BALANCE &
>      IF TYPE OF ACCOUNT-DETAIL = "INVOICE" OR &
>         TYPE OF ACCOUNT-DETAIL = "INTEREST"
>SUM AMOUNT OF ACCOUNT-DETAIL INTO INVOICE-BALANCE NEGATIVE &
>      IF TYPE OF ACCOUNT-DETAIL = "PAYMENT"
>FILE ACCOUNT-DETAIL ALIAS INTEREST ADD AT INVOICE-NO &
>      IF DAYS(SYSDATE)  > DAYS(INVOICE-DATE) + 30
>  ITEM AMOUNT FINAL INVOICE-BALANCE * 0.01
>  ITEM TYPE FINAL "INTEREST"
>FILE ACCOUNT-MASTER UPDATE AT ACCOUNT-NO
>SUM AMOUNT OF INTEREST INTO BALANCE OF ACCOUNT-MASTER
>GO
```

The account details are accessed and sorted on account number and invoice number.

Two temporary items, INVOICE-DATE and INVOICE-BALANCE are created to hold the date and accumulated outstanding balance of each invoice.

The two SUM statements accumulate the outstanding balance.

The first FILE statement together with the following two ITEM statements create a new detail record for the interest if the invoice is past due.

The last FILE statement and following SUM statement update the account balance to reflect the new interest change.

2. Standard Batched Update

The standard batch update is probably the most universal QTP application. At the end of each day, a company wants to total all money received and prepare for the next day's transactions. With QTP, this assignment could be performed in ten specification lines.

```
>ACCESS BATCH-HEADER LINK TO TRANS
>SELECT IF TOTAL-ENTERED = TOTAL-CALCULATED
>SORT ON ACCOUNT-NO
>FILE ACCOUNT-DETAIL ADD
>  ITEM TYPE INITIAL "PAYMENT"
>FILE ACCOUNT-MASTER UPDATE AT ACCOUNT-NO
>SUM AMOUNT OF TRANS INTO BALANCE OF ACCOUNT-MASTER
>FILE BATCH-HEADER DELETE
>FILE TRANS DELETE
>GO
```

The ACCESS, SELECT and SORT statements retrieve transactions from balanced batches and sort them by account number.

The FILE statement for ACCOUNT-DETAIL creates payment records from the transactions.

The FILE statement together with the following SUM statement update ACCOUNT-DETAIL to reflect the new payments.

The final two FILE statements delete all processed batches and transactions.

TRANSACTION PROCESSOR STATEMENTS

The major specification statements used in QTP will be as follows:

ACCESS: specifies the files to be read, the order in which they should be read and linkage between files.

BUILD: takes all requests defined up to the BUILD statement and saves these requests into a named MPE file for future use.

CHOOSE: specifies an explicit set of data by key for retrieval.

DEFINE: used to define a frequently used expression.

EDIT: specifies that input files be edited according to editing defined in QSCHEMA.

FILE: defines an output action to be performed on a file. These actions are:

ADD add if record does not exist
UPDATE add if record does not exist, else replace
REPLACE replace if record exists
DELETE delete if record exists.

ITEM: indicates specific items to be assigned initial or final values, or to accumulate totals.

RESET: resets status control options to original status.

SELECT: restricts selection of records for processing to those which satisfy a condition.

SORT: specifies the order in which records are sorted.

SUBFILE: creates a sequential file (MPE).

Does not require the file to exist in the QSCHEMA. Will produce its own schema information in the header of the file, and be accessible to both QTP and QUIZ.

TEMPORARY: creates a temporary-item which does not exist in the database.

DESIGN CONSIDERATIONS

To arrive at a smoothly functioning product, certain design considerations were uppermost in the thoughts of the development team.

- The desirability of a specification based language to insulate users from procedural constructs.
- The need to support complicated production runs as well as ad-hoc file maintenance functions.
- The need for efficient run time performance. Since QTP will run repetitively against bulk volumes of data, its design will differ significantly from a data entry system which requires a high degree of user interaction.
- The need for effective interaction with QUICK to allow class data changes in conjunction with data entry.
- The automatic insulation of users from migrating secondary and other file positioning problems inherent in IMAGE and KSAM file updates.

SUMMARY

Application systems on the HP3000 are typically composed of numerous data entry screens, numerous reports and a relatively small number of batch processes which run on a regular basis at day-end, month-end and year-end. Significant progress has been made towards eliminating the need to custom program data entry and reporting functions. Very little attention has been paid to the development of productivity tools to perform standard batch processing functions. The transaction processor is designed to perform most standard batch operations as well as a wide range of file manipulation functions. QTP, together with its companion products QUIZ, QUICK, and QSCHEMA, will form a complete application generator for the HP3000. Complete systems can be built using these components with major savings in programmer resources applied to development and maintenance, and with real gains in data integrity, system consistency and flexibility.

RISE
An RPG Interactive System Environment for Program Development

Gary Ow
Hewlett-Packard
Information Networks Division

I. ABSTRACT

RISE is a specialized editor dedicated to the creation and modification of programs written in RPG. Its ultimate purpose is to significantly increase RPG programmers' productivity by presenting a single, friendly user interface to a development environment in which a user may easily edit source code as well as compile, prepare, and run the program. This paper will discuss some of the key features of RISE such as the visual forms that represent the RPG Specification Record forms and direct screen editing.

II. INTRODUCTION

RPG is a computer programming language in a class of its own. Some programmers boast of never using it, others are embarrassed to admit using it, while there are those individuals who are proud of using it. Whatever personal opinion these programmers may have, they all must admit that RPG can very effectively accomplish their data processing tasks no matter how complex.

RPG has proven to be extremely successful in the commerical data processing market. It is a higher-level specification language and, as such, it is very practical and easy to implement applications quickly. Simple notation provides easy management of complex I/O operations. Specifically, Hewlett-Packard's RPG/3000 offers automatic interfaces to Image, V/3000, and KSAM which are often crucial elements in a commerical application.

Although the language appears to be perfect for commercial processing, it has one major flaw . . . its column dependent syntax leads to tremendous difficulties during program creation and modification.

RPG programmers are all too fimiliar with these difficulties. But no longer must they fight these problems, sitting through long, tedious program development sessions, because now they have RISE.

RISE presents RPG programmers with a single working environment in which they may easily edit RPG source code using special forms that resemble the RPG Specification Record forms to alleviate column counting and confusion. Furthermore, RISE allows users to compile, prepare, and run programs as well as execute many MPE commands and execute any program file. RISE enhances the power of RPG, eliminating RPG's syntactic flaw, providing a friendly user interface. RISE is totally dedicated to RPG users as illustrated in the next section.

III. RISE

The RPG Interactive System Environment offers these features:

- Visual editing with images that represent the RPG Specification Record forms.
- Command menu and special function keys (softkeys) for a broad range of uses.
- Page-at-a-time direct screen editing.
- Ability to call the RPG compiler as well as the Segmenter and manipulate the compiled listing using a split screen.
- Choice of editing a file directly or editing a copy of that file.
- Ability of execute MPE commands and run any program file.
- Help facility.
- Ability to create a comment banner in which a user may type documentation.
- Renumbering command which also allows a user to renumber the lines of the RPG source program in columns 1-5.
- Add mode which allows a user to change RPG Specification Record forms, and to DELETE, LIST, or MODIFY lines while in Add mode.
- Recoverability feature to restore deleted lines.
- Automatic instructive guided tour to quickly acquaint a new user with the system.

Together, these features have been carefully integrated into RISE, resulting in a productivity tool essential to all RPG programmers.

Friendliness and powerful capabilities were guiding principles in design. The following example illustrates one of RISE's friendly aspects. When RISE detects an error in the user's entered command, it tries to return an error message which describes what exactly is wrong and possibly how to correct it. For instance, if a user entered just the letter "C", RISE will respond "Can't

distinquish CHange, COMment, or COPy command". This message is far more informative then the usual "Unrecognized command" message.

The following is a summary of all commands which RISE offers:

1. ADD —add new lines
2. BEGIN —start editing a new file
3. CHANGE —change oldstring to newstring
4. COMMENT —create a comment banner
5. COPY —duplicate lines
6. DELETE —delete lines
7. EXIT —end system
8. FILE —edit file directly
9. FIND —locate a string
10. FORM —display an RPG Specification Record form
11. GET —execute commands from a file
12. HELP —explain commands
13. INCR —set default increment value
14. JOIN —append or merge a file
15. KEEP —save the work file
16. LINE —enter Line Mode
17. LIST —list lines
18. MENU —display Command Menu
19. MODIFY —modify lines
20. MOVE —transfer lines to new location
21. PRINT —print lines offline formatted
22. RENUM —renumber editor sequence numbers or RPG source line numbers
23. RUN —run a program file
24. SHOW —display a page for direct screen editing
25. TEXT —edit copy of a file
26. UNDEL —restore last deleted lines
27. VERIFY —compilation or preparation
28. XPAND —expand the work file size
29. :MPEcommand —execute MPE command

However extensive a user's experience is, however technical the job at hand, with this full command set a user may easily direct RISE to accomplish editing and data processing goals.

IV. KEY FEATURES OF RISE

RISE offers a multitude of features dedicated to the development and maintenance of RPG source code. In this section, a presentation on some of RISE's major features is given.

RPG Specification Record Forms

Part of RISE's user interface consists of a communication system which utilizes special forms that are equivalent to the "RPG Listing Analyzer." Using graphic guides, these forms explain the semantics of every column so that the RPG source code is understandable as well as easy to edit. This eliminates confusion and extensive column counting.

When creating new source lines, a user may direct RISE to display any of the RPG Specification forms, and when modifying existing lines, RISE will automatically display the appropriate form with the line for modification.

Add Mode

RISE offers an Add Mode similar to Edit/3000 to allow a user to enter new text. However, RISE's Add Mode is significantly spiced up to more thoroughly meet the user's needs of source code creation.

While in Add Mode, a user may perform any of the following operations:

1. Change RPG Specification Record forms
2. Display a column indicator form
3. Enter the DELETE, LIST, or MODIFY command
4. Direct RISE to prompt for new text starting at column 6 instead of 1 to skip over the entering of the optional RPG line numbers in columns 1-5

The ability to enter the DELETE, LIST, or MODIFY command while in Add Mode is often very valuable and time-saving. For instance, suppose a user is entering new source lines and discovers that the previous line was entered incorrectly. No problem . . . for the user may immediately access the line and modify it and simply continue on with the additions. This is all accomplished without the user switching out and back into Add Mode which can be quite bothersome.

Modify Mode

Moreover, RISE offers a Modify Mode for interactive line editing similar to Edit/3000 but with added improvements. The first improvement is that the record to be modified can be automatically displayed with its associated RPG Specification Record form for guidance while editing. Another is a recovery feature which allows a user to type Control-Y to restore the record in the form before any modifications were made to it. Finally, the last improvement was implemented for visual consistency. Unlike Edit/3000, when RISE displays a record for modification, it appears formatted on the screen exactly as records created in Add Mode and records listed with the "LIST" command appear. In otherwords, the line number will always appear to the left of a record whenever a user creates it, modifies it, or lists it.

Show Mode

RISE's Show Mode displays an RPG Specification Record form at the top of the screen followed by a page of RPG source code in Block Mode for direct screen editing. A user need only use the terminal's cursor control keys to position the cursor on the page and directly type in the changes. The user could also depress the Tab key to quickly skip across the source record to important columns of interest.

With this feature, the editing procedure is tremendously simplified for what is visible on the screen will

duplicate what is stored in the file so that "what you see is what you'll get."

The special function keys or softkeys are also integrated into Show Mode, giving the user even more flexibility. Descriptive softkey labels appear at the top of the screen to inform the user of the function of each softkey. Some of the functions performed by the softkeys are variations on scrolling such as Scroll Forwards, Scroll Backwards, Scroll to First Page, and Scroll to Last Page.

With this scrolling power, a user could page through the file, making any necessary changes directly on the screen. The RPG Specification Record form will automatically change on the screen whenever the next form type changes.

Help Facility

Like all high quality, interactive software systems, RISE includes a Help Facility which provides a summary of all commands and detail descriptions of each command. Each detail description shows a command's syntax, explains its operation, and gives examples.

Command Menu

Besides entering commands to RISE based on syntax rules, a user may select the next command to be executed with a "Command Menu." The Command Menu displays the options for a command so that a "fill out the blanks" on the menu to express the next command, ignoring the syntax rules. This is added convenience since the menu also functions as an implicit help guide by documenting the parameters of a command, proving beneficial to new users.

Some Useful Commands

Other than just the normally expected editing commands supported in standard editors such as the COPY, MOVE, FIND, and CHANGE commands, RISE offers three additional commands designed for user convenience and deletion recovery.

The COMMENT command will create a comment banner composed of a rectangle of asterisks to be placed anywhere in the user's source code. The banner is displayed in Show Mode, allowing a user to directly type in documentation on the screen within the banner. Because this feature makes documentation of source code less troublesome, it will encourage programmers to more thoroughly perform this task.

The RENUM command not only renumbers the sequence numbers of the file being edited, but it also can renumber the RPG line numbers of the source code in columns 1-5 so that a user never has to type in the numbers.

Finally, the UNDEL command will restore all lines that were deleted by the last DELETE command, protecting a user from accidental or erroneous deletions. This feature, at times, can save many hours of work lost due to incorrect editing actions.

Compilation, Preparation, and Execution

Within the RPG Interactive System Environment, a user may compile, prepare, and execute the program to ensure its compile-time and run-time correctness. With this ability to invoke the RPG compiler, a user could immediately catch all compilation errors without having to switch back and forth between the operating system and RISE.

After compiling a user's program, RISE will automatically manage the compilation listing so that a hard copy listing need not be printed. By pressing softkeys, a user could scroll through the listing at the terminal, page by page, for inspection. Moreover, RISE has a unique capability to allow a user to view the listing in a split-screen mode, displaying two different portions of the listing at once. By pressing a softkey, a user could split the screen into a top and bottom portion and scroll either. Essentially, a user could simultaneously have two "windows" to view the compilation listing. In this fashion, a user may display the compiled source code in the top window and the compiler generated error messages in the bottom window.

Actually, RISE can do this automatically for a user. By simply depressing the "Find Error" softkey, a user can direct RISE to automatically locate the next source line in error and display it in the top window along with its corresponding error message displayed in the bottom window. Using this powerful feature, a user could quickly locate problems in the source code and correct them all while in the same development environment.

After developing the RPG program to compile successfully into a USL file, the user may command RISE to prepare the USL file into a program file. To accomplish the preparation, RISE invokes the Segmenter.

At this stage, before executing the program file with RISE's RUN command, the user may issue any file equations necessary for the proper runtime execution of the program using RISE's ":MPEcommand". Finally, the user is all set to execute the program.

Incidentally, a user may execute any program file within RISE such as the system utilities FCOPY and SPOOK.

RISETOUR

Another instructive feature is RISETOUR which is a self-paced, interactive, guided tour of RISE most beneficial to new users. Generally, it is always very difficult for a new user to become accustomed with a new interactive software package. However, in the case of RISE, a user could sit down at a terminal and RISETOUR will guide the user through the major features of RISE, giving demonstrations, explanations, and diagrams. This gives a new user first hand experience with a new product, easing the learning process so that the user may become competent with it more quickly.

V. CONCLUSION

RISE presents RPG programmers with a single, coherent program development environment dedicated to their production needs. The RPG Specification Record forms allow users to edit their source code in a friendly and easy manner. Direct screen editing further simplifies the editing procedure. The ability to generate comment banners encourages more thorough documentation of the source code. An online Help Facility, Command Menu, and RISETOUR all effectively acquaint new users to the system. The ability to compile the program with automatic management of the listing in split-screen mode detects errors immediately for correction. And finally, the ability to issue many MPE commands as well as execute any program file is a passage way to other utilities from RISE.

Together, these features and many more have been smoothly integrated into a productivity tool vital to all RPG programmers. They can finally "rise" out of their confusing, tedious, and time-consuming development problems using the RPG Interactive System Environment.

IMAGE/COBOL: Practical Guidelines

David J. Greer
Robelle Consulting Ltd.
Aldergrove, B.C., Canada

SUMMARY

This document presents a set of practical "rules" for designing, accessing, and maintaining IMAGE databases in the COBOL environment. This document is designed to aid systems analysts, especially ones who are new to the HP3000, in producing "good" IMAGE database designs. Each "rule" is demonstrated with examples and instructions for applying it. Attention is paid to those details that make using the database trouble-free for the COBOL programmer, and maintaining the database easier for the database administrator.

CONTENTS

Copyright 1981. All rights reserved.
Permission is granted to reprint this document (but not for profit), provided that Copyright notice is given.
This document was prepared with Prose, a text formatter distributed with software to all Robelle customers.

DATABASE DESIGN

IMAGE/3000 is the database system supplied by Hewlett-Packard;[6] it is used to store and retrieve application information. A database does not suddenly appear out of thin air; it develops through a long and involved process. At some time, a logical database design must be translated into the actual schema that implements a physical IMAGE database. This phase is the most difficult of the database development cycle.[7] The IMAGE/3000 Reference Manual[6] contains a sample database called STORE, which demonstrates most of the attributes of IMAGE. Throughout this document, the STORE database will be referenced when examples are needed.

Logical Database Design

The foundation of a new database is a logical design, which is created by examining the user requirements for input forms, for on-line enquiries, and for batch reports. The database should be viewed as an intermediate storage area for the information that comes from the input forms and is eventually displayed on the output reports.[9] [10]

Database design is normally done from the bottom up, as opposed to structured program design, which is usually done from the top down. The starting point for a database is the elements (items) that will be stored in the database. These data elements represent the user's information. In the early stages, the size and type of these elements are not needed, only the name and values.

Rule: *Start your logical database design by naming each data item, then identify what values it can have and where it will be used.*

Here is an example of a subset of data items for the STORE database:

```
CUST-STATUS      Two characters, attached to each customer record.
                     Valid values are: 10=advanced, 20=current,
                     30=arrears and 40=inactive.

DELIV-DATE       Six numeric characters; Date, YYMMDD, attached to
                     every sales order as the promised delivery date.

ON-HAND-QTY      Seven numeric characters, attached to every inventory
                     record to show the current quantity of an
                     inventory item available for shipping.

PRODUCT-PRICE    Eight numeric characters, (6 whole digits, 2 decimal
                     places), attached to every sales record.  This is
                     the price of a product sold, on the date that
                     the sale was made.
```

As the logical database design develops to deeper levels of detail, the elements needed should eventually reach a stable list. These elements should then be combined into records by grouping logically related items together.

It is important that "repetition" be recognized early in the design. An example of this is a customer's address. The most flexible method of implementing addresses is a variable number of records associated with the customer account number. Another method is to make the address field an X-type variable (e.g., X24) repeated 5 times (e.g., 5x24). Repeated items are often the most natural way to represent the user's data, so the use of repeated items is encouraged.

After the records are designed, enquiry paths must be assigned. During the early stages of database design, it is important to use elements that are readable and easy to implement with the tools at hand. This permits testing of the database using tools such as QUERY, AQ, and PROTOS.

Physical Database Design

After the local database is designed, the IMAGE schema must be developed. The restrictions of IMAGE must now be worked into the database design.

IMAGE requires that all items needed in the database be defined at the beginning of the schema, and a size and type must be associated with each. Initially, declare each item as type X (display); later, the data type may be altered.

Records are implemented as IMAGE datasets. Start by treating each record format as a master dataset.

Rule: *If a record is uniquely identified by a single key value, start by making it a master dataset (e.g., customer master record keyed by a unique customer number).*

The STORE database assumes that each CUST-ACCOUNT field is unique. Furthermore, there is only one customer record for each CUST-ACCOUNT. All of the information describing one customer is gathered together to result in the M-CUSTOMER dataset:

```
NAME:  M-CUSTOMER,        MANUAL (1/2);        <<PREFIX = MCS>>
ENTRY:
       CITY
      ,CREDIT-RATING
      ,CUST-ACCOUNT(1)       <<KEY FIELD>>
      ,CUST-STATUS
      ,NAME-FIRST
      ,NAME-LAST
      ,STATE-CODE
      ,STREET-ADDRESS
      ,ZIP-CODE
      ;
CAPACITY:  211;      <<M-CUSTOMER,PRIME; ESTIMATED>>
```

Rule: *If a "natural" master dataset will require on-line retrieval via an alternate key, drop it down to a detail dataset.*

The detail dataset will have two keys: the key field of the original master dataset, and the alternate key. You will have to create a new automatic master for the original key field, and you may have to create an automatic master for the alternate key (unless you already have a manual master dataset for that item).

Take the M-CUSTOMER dataset as an example. Assume that in addition to needing on-line access by CUST-ACCOUNT, it is also necessary to have on-line access by NAME-LAST. The following dataset structure would result:

```
NAME:  A-CUSTOMER,        AUTOMATIC (1/2),     <<PREFIX = ACS>>
ENTRY:
       CUST-ACCOUNT(2)      <<KEY FIELD>>
      ;
CAPACITY:  211;      <<A-CUSTOMER,PRIME; ESTIMATED>>

NAME:  A-NAME-LAST,       AUTOMATIC (1/2),     <<PREFIX = ANL>>
ENTRY:
       NAME-LAST(1)      <<KEY FIELD>>
      ;
CAPACITY:  211;      <<A-NAME-LAST,PRIME; CAP(A-CUSTOMER)>>

NAME:  D-CUSTOMER,        DETAIL (1/2);        <<PREFIX = DCS>>
```

```
ENTRY:
        CITY
       ,CREDIT-RATING
       ,CUST-ACCOUNT(!A-CUSTOMER)      <<KEY FIELD, PRIMARY>>
       ,CUST-STATUS
       ,NAME-FIRST
       ,NAME-LAST(A-NAME-LAST)         <<KEY FIELD>>
       ,STATE-CODE
       ,STREET-ADDRESS
       ,ZIP-CODE
       ;
CAPACITY:  210;      <<D-CUSTOMER; CAP(A-CUSTOMER)>>
```

Rule: *If an entry can occur several times for the primary key value, store it in a detail dataset.*

Detail datasets are for repetition and multiple keys. Master datasets can only contain one entry per unique key value. An example of repetition in a detail dataset is a customer address field. The customer address can be stored as a repeated field in a master dataset, but eventually there will be an address that will not fit into the fixed-size repeated field. Instead of a repeated field, use a detail dataset to store multiple lines of an address. For example:

```
ADDRESS-LINE,          X24;     << An individual line of address.  This
                                   item is used in D-ADDRESS to provide an
                                   arbitrary number of address lines for
                                   each customer.
                                >>
CUST-ACCOUNT,          Z8;      << Customer account number.  This field
                                   is used as a key to the M-CUSTOMER
                                   and D-ADDRESS datasets.
                                >>
LINE-NO,               X2;      << Used to keep address lines in D-ADDRESS
                                   in the correct order.  This field also
                                   provides a unique way of identifying
                                   each address line for every
                                   CUSTOMER-ACCOUNT.
                                >>

NAME:  D-ADDRESS          DETAIL (1/2);       <<PREFIX = DAD>>
ENTRY:
        ADDRESS-LINE
       ,CUST-ACCOUNT(!M-CUSTOMER(LINE-NO))  <<KEY FIELD, PRIMARY PATH>>
       ,LINE-NO                             <<SORT FIELD>>
       ;
CAPACITY:  844;      <<D-ADDRESS; 4 * CAP(M-CUSTOMER)>>
```


Dataset Paths

The following definition of PATHs and CHAINs comes from Alfredo Rego:[11]

> A PATH is a relationship between a MASTER dataset and a DETAIL dataset. The master and the detail must contain a field of the same type and size as a common "bond," called the SEARCH FIELD. A path is a structural property of a database.
>
> A CHAIN, on the other hand, contains a MASTER ENTRY and its associated DETAIL ENTRIES (if any), as defined by the PATH relationship between the master and the detail for the particular DETAIL SEARCH FIELD. . . . A chain is nothing more than a collection of related entries (for instance, a bank customer would be the master entry and all of this customer's checks would be the detail entries; the "chain" would include the master AND all its details; the chain for customer number 1 would be completely different from the chain for customer number 2).

Paths provide fast access at a certain cost: adding and deleting records on the path is expensive. The more paths there are, the more expensive it gets.[1] Another restriction of paths is that there can be a maximum of 64,000 records on a single path for a single key value. This sounds like a large number, but it can be very easy to expand a chain to this size if a key is specified for a specific, reporting summary program (e.g., billing cycle, in monthly billing transactions).

Rule: *Avoid more than two paths into a detail dataset.*

There are some instances where three paths are necessary, but these should be avoided as much as possible. Before adding a path, examine how the path is going to be used. If it is added just to make one or two batch programs easier to program, the path is not justified. The batch programs should serially read and sort the dataset, then merge the sorted dataset with any other necessary information from the database.

The date paths of the SALES dataset of the STORE database are good examples of unnecessary paths. Because the chain lengths of paths organized by date are almost always very long, such a chain is rarely allowed. Also, users are often interested in a large range of dates (such as a month, quarter or year), not just a specific day.

In order to obtain the same type of reporting by date, it is possible to do one of the following: (1) read the database every night and produce a report of all records entered every day; (2) keep a sequential file of all records added to the dataset on a particular day. This file can then be used as an index into the database.

These are not the only solutions to removing the date paths, but they indicate the kind of solutions that are possible. Because of the high volume and length of the average chain, date paths are prime candidates for removal from a database.

The following example demonstrates how the SALES dataset should have been declared:

```
<<  The D-SALES dataset gathers all of the sales records
    for each customer.  The primary on-line access is by customer,
    but it is necessary to have available the product sales
    records.  The PRODUCT-PRICE is the price at the time
    the product is ordered.  The SALES-TAX is computed based
    on the rate in effect on the DELIV-DATE.
>>

NAME:  D-SALES,          DETAIL (1/2);       <<PREFIX = DSA>>
ENTRY:
       CUST-ACCOUNT(!M-CUSTOMER)   <<KEY FIELD, PRIMARY PATH>>
      ,DELIV-DATE
      ,PRODUCT-NO(M-PRODUCT)       <<KEY FIELD>>
      ,PRODUCT-PRICE
      ,PURCH-DATE
      ,SALES-QTY
      ,SALES-TAX
      ,SALES-TOTAL
      ;
CAPACITY:  600;     <<D-SALES; 3 * CAP(M-CUSTOMER)>>
```

Rule: *Avoid sorted paths.*

Because sorted paths can require very high overhead when records are added or deleted, they should be avoided as much as possible. There are some instances when a sorted path makes the system and program design much easier, but this convenience must be traded off against the highest cost of maintaining sorted chains.

The most important criteria in evaluating sorted chains are: (1) whether the chain is needed for batch or on-line access. In batch, it is possible to read and sort the dataset, rather than relying on sorted chains. In an on-line program, this is usually not possible, so sorted chains are required. (2) How long is the average chain going to be? The longer the chain, the more expensive it is to keep sorted. If chains have fewer than 10 entries per key value on average, sorted chains can be permitted. (3) How are records being added to the dataset? If a sorted chain is present, and data is added to the dataset in sorted order, there is very little extra overhead in the sorted chain. If, on the other hand, data is added in random fashion, there is a very high cost associated with the sorted chain.[11–13]

Locking Strategy

Early in the database design, it is important to identify the locking necessary for the application. The easiest choice is to use database locking. Unless specific entries are going to be modified by many users, database locking should work. Remember: locking is only needed when updating, adding, or deleting entries from the database, not when reading entries. Never leave the database locked when interacting with the terminal user.

The next level of locking to be considered is dataset locking. This takes more programming, but provides for a more flexible locking strategy.

Rule: *Never permit MR capability to programmers;* instead, use lock descriptors (and a single call to DBLOCK) to lock all datasets needed.

For very complicated systems (e.g., an inventory system with inventory levels that must be continually updated), record locking should be used. The database design should help the application programmer by making the easiest possible locking strategy available for each program.[2]

Passwords

Most application systems go overboard in their use of database passwords. The simplest scheme to implement is a two-password system. The database is declared with one password for reading and one for writing. Each password is applied at the dataset level; and item-level passwords are not used.

Rule: *Use the simplest password scheme that does not violate the database integrity.*

The advantages to this scheme are that there are fewer passwords to remember, IMAGE is more efficient (because all security checks are done at the dataset level, instead of the data item level), and the user can still use tools such as QUERY, by being allowed the read-only password.

In sensitive applications, a separate dataset or database can be used to isolate data requiring special security. This still permits the simplest password scheme possible, with an extra level of security. The following example shows how to declare passwords for read-only access and read/write access on a dataset level:

```
PASSWORDS:      1 READER;
                2 WRITER;
```

The declarations for the M-CUSTOMER and the D-SALES datasets contain "(½)" on the line that declares the name of the datasets. The "(½)" indicates that the READER and WRITER passwords are in effect for the whole dataset.

Early Database Testing

The early database design should allow the user or analyst to experiment on the database design with test data. User tools such as QUERY or AQ should be used to access the database. At this stage, the item types may be left approximate, so long as the user or analyst gets a chance to interact with the database design. The analyst should check that all requirements of the user can be met by the database design.

Rule: *Build your test databases early. Use an application tool to verify that the database design is correct.*

In some cases, the end user may not be able to access the database, but the database designer must go through this testing process. This examination of the database design may uncover design flaws which can be fixed easily at this early stage. After the logical database design has been roughly packaged as an actual IMAGE database and verified against the user requirements, the design should be optimized and the finishing touches added (see next section).

Very Complex Databases

IMAGE has a number of size restrictions that it imposes on the database design. For example, the number of items in a database is limited to 255, and the number of datasets in a database is limited to 99. For many applications, these limits pose no problems; but with the larger databases being designed today, it is not difficult to imagine databases which exceed these lmits. What can you do to get around this problem?

Bottom-Up Design

The design method outlined above must be extended. For small projects, it is adequate to simply group related data items into datasets, because the entire application will fit into one database. However, for large projects, another step is required: related datasets must be grouped into separate databases.

Multiple databases introduce new problems for the application programmer. These include larger programs, which result in larger data stacks, as well as problems with locking. In designing a multiple database system, it is best to minimize the number of programs that must use more than one database.

If an application decomposes into independent subunits, few programs will require more than one database. The design of the system and the database may have to be revised to increase the independence of the sub-systems.

POLISHING DATABASE DESIGN

The database designer has two main concerns in completing the database design. Will the application programs be able to access the database within the defined limits of the HP3000? Does the database take best advantage of COBOL and other tools available?[3-8-11-13]

Overall Performance

Rule: *Always make a formal estimate of on-line response times and elapsed times for batch jobs.* If the project is going to require additional hardware resources, it is better to know it before the project goes into production.

The following material is taken from *On Line System*

Design and Development,[9] with comments and examples to expand on the original. The HP3000 is able to perform approximately 30 I/Os per second. On various machines under different operating systems, it may be possible to obtain more than this. Because it is extremely difficult to obtain the theoretical maximum of 30 I/Os per second, it is best to plan for a maximum of 20 I/Os per second.

Each IMAGE procedure results in a specific amount of I/O. Before going ahead with a large application, the total I/O required for the application must be computed and compared against the maximum. This is done by estimating the I/O for each on-line function, then summing the I/Os of the functions that might reasonably occur concurrently. Also, the total elapsed time for batch jobs must be estimated to ensure that they will complete in the time available.

The following gives an approximate measure of the number of I/Os necessary for each IMAGE procedure in an on-line environment:

Procedure	I/O
DBGET	1
DBFIND	1
DBLOCK	0
DBUNLOCK	0
DBUPDATE	1
DBPUT	2 + 2 * Number of keys in the dataset.
DBDELETE	2 + 2 * Number of keys in the dataset.

The figures for DBPUT and DBDELETE do not take into account sorted chains. If sorted chains are kept short, the above figures will work. If sorted chains are long, the following formula gives an approximate measure of how many I/Os are required to add records in random fashion to a sorted chain:

```
2 + 2 * number of keys + (average chain length / 2)
```

All of the above figures for the number of I/Os for each IMAGE procedure are the same in batch, with one exception. If a batch program reads a dataset serially, the I/Os required will be:

```
Serial DBGET I/Os = number of records / blocking factor
```

If the batch program also does a sort of all of the selected records from the serial DBGET, the number of I/Os will be increased.

The following example computes how long a specific batch program will take to run; the program makes the following IMAGE calls:

Batch Calculation Example

```
125,000 DBGETs serial; blocking factor is 5.
 80,000 DBPUTs to a detail dataset with two keys.
 80,000 DBFINDs.
 80,000 DBGETs to the dataset with the DBFIND.
 80,000 DBUPDATEs.

 Total I/Os required =

 I/Os for DBGET  (205,000 / 5) plus
 I/Os for DBFIND ( 80,000 X 1) plus
 I/Os for DBPUT  ( 80,000 X 6) plus
 I/Os for DBUPDATE(80,000 X 1).

 equals   681,000 I/Os.

 We can do approximately 20 I/Os a second so

      681,000
      -------    =  34,050 seconds  =  9.5 hours
         20
```

If the batch program also is intended to run overnight, but is unlikely to finish in one evening, because time is also needed for backup and other daily functions.

Improving Performance

How can the total time of this program example be reduced to 3.9 hours? One way is to replace the DBPUT with a DBUPDATE. In many instances it is possible, through changes in the application and database design, to use a DBUPDATE instead of a DBPUT. This is especially true in environments where there are recurring monthly charges, which change only slowly over time.

There is another advantage to using DBUPDATE. For each DBPUT, a record is added to the database, and this record must later be deleted using DBDELETE. Because it takes as long to delete the record as it did to add it in the first place, the DBUPDATE can provide as much as an eight-fold decrease in running time, compared with DBPUT/DBDELETE.

COBOL Compatibility

When designing a database, keep in mind how the database is going to be used (COBOL, QUERY, AQ, PROTOS, etc.). The following rules apply to item types and should be used throughout the database.

Numeric Fields

When the database was first designed, all fields were initially declared as type X (display). By now you should know the likely maximum value for each data item. Once the size of each data item is fixed, the time has come to specify a more efficient data type for numeric fields.

The type of field used for numeric values depends on the maximum size of the number to be stored (i.e., the number of digits, ignoring the sign). The following table should be used in determining numeric types:

Number of Digits	IMAGE Data type
<5	J1
<10	J2
>=10	Packed-decimal of the appropiate size.

Rule: *For numeric fields, use J1 for fewer than five digits; use J2 for fewer than ten digits; otherwise, use a P-field (packed-decimal) of the appropriate size.*

In COBOL, an S9(2)V9(2) COMP variable is considered to have a size of 4, or J1. The one exception to this rule is sort fields. All sort fields must be type X. If a numeric sort field is required, it must be declared as type X and redefined as zoned in all COBOL programs. Remember that packed fields in IMAGE are always declared one digit larger than the corresponding COBOL picture (S9(11) COMP-3 becomes P12) and must be allocated in multiples of four.

COBOL databases must not contain R-fields, because R-fields have no meaning in the COBOL language. Instead of an R-type field, a J-type or P-type field must be used. The STORE database contains an R-type field, CREDIT-RATING, which should have been declared as:

```
CREDIT-RATING,          J2;     << Customer credit rating.  The larger
                                   the number, the better the customer's
                                   credit.  Used to five decimal places.
                                >>
```

Key Types

Every key, whether in a master or detail dataset, must be hashed to obtain the actual data associated with the key value. Hashing is a method where a key value, such as customer number 100, is turned into an address. The method used tries to generate a different address for every key value, but in practice this is never possible. The choice of the type of key has a large bearing on how well the hashing function will work.

Rule: *Always use X-type, U-type, or Z-type keys, and never use J-type, R-type, P-type, or I-type keys.* Type X, type U, and type Z keys give the best hashing results.

When using a Z-type for a key, leave it as unsigned in all COBOL programs. Because key values rarely have negative values, there is no effect on the application by removing the sign from a zoned field. The advantages to leaving off the sign are: (1) displaying the field in COBOL or QUERY results in a more "natural" number, and (2) problems between positive, signed, and unsigned zoned numbers are avoided.

Date Fields

Rule: *Dates must be stored as J2 (S9(6) COMP) in YYMMDD format.*

This format provides the fastest access time in COBOL and takes the least amount of storage. Use a

standard date-editing routine to convert from internal to external format and vice versa.[4]

The only exception to this is when a detail chain must be sorted by a date field. Because IMAGE does not allow sorting on J2 fields, X6 is used. For the chain to be sorted correctly, the date must still be stored in YYMMDD format.

Other Item Types

The only item types that should be used are J- or P-types for numeric values, and X-, U- or Z-types for keys. The K-, I- and R-types should never be used in a commercial application where COBOL is the primary development language.

Example

Earlier, in the discussion of logical database design, four items were described: CUST-STATUS, DELIV-DATE, ON-HAND-QTY, and PRODUCT-PRICE. The following example gives the actual IMAGE declaration for each of these items, according to the rules of this section:

```
CUST-STATUS,          X2;    << Defined state of a particular customer
                                account.  The valid states are:
                                10 = advance
                                20 = current
                                30 = arrears
                                40 = inactive
                             >>
DELIV-DATE,           J2;    << Promised delivery date.
                             >>
ON-HAND-QTY,          J2;    << Amount of a specific product currently
                                onhand.  Only updated upon
                                confirmation of an order.
                             >>
PRODUCT-PRICE,        J2;    << Individual product price, including
                                two decimal points.
                             >>
```

Primary Paths

Rule: *Assign a primary path to every detail dataset.*

IMAGE organizes the database so that accesses along the primary path are more efficient than along other paths. The primary path should be the path that is accessed most often in the dataset.

If there is only one path in a detail dataset, it must be the primary path. If there are two paths that are accessed equally often, but one is used mostly in on-line programs and the other mostly in batch programs, assign the primary path to the one that is used in on-line programs. A primary path is indicated by an exclamation point (!) before the dataset name that defines the path. A path with only one entry per chain should not be selected as a primary path.

The Schema

The IMAGE schema is the method by which you tell both IMAGE and the programmers what the database looks like. The schema should be designed with maximum clarity for the programmer, because IMAGE is only partly concerned with the schema's layout.

Rule: *The schema file name is always XXXXXX00, where XXXXXX is the name of the database.*

This naming convention makes locating the schema easier for all staff. The file is always located in the same group and account as the database. If the database name was STORE and the STORE database was built in the DB group of the USER account, the schema name would be STORE00.DB.USER.

Layout

A clear layout of the schema makes the programmer's job easier. Some requirements of the layout are imposed by IMAGE, but there are still a number of things that the database designer can do to make the schema more understandable.

Every database schema should start with a $CONTROL line. The $CONTROL line must always contain the TABLE and BLOCKMAX parameters. The default BLOCKMAX size of 512 should always be used when first implementing the database. Later, after careful consideration, the BLOCKMAX size may be changed. When first designing the database, $CONTROL NOROOT should be used.

The $CONTROL line should be followed by the name of the database. This is followed by a header comment. This comment describes the designer of the database, the date, the conventions used in designing the schema, abbreviations that are used within IMAGE names, and sub-systems with which the database is compatible and incompatible.

The following are the opening lines of the example STORE database:

```
$CONTROL TABLE,BLOCKMAX=512,LIST,NOROOT
BEGIN

DATA BASE   STORE;

<<                  STORE DATABASE FROM THE IMAGE MANUAL

AUTHOR:     DAVID J. GREER, ROBELLE CONSULTING LTD.

DATE:       DECEMBER 15, 1981

CONVENTIONS:

This schema is organized in alphabetic order.  All master datasets are
listed before detail datasets, and automatic masters come before
manual master datasets.

All dates are stored as J2, YYMMDD, except where they are used as
sort fields.  If a date is a sort field, it is stored as X6, YYMMDD.

The following abbreviations are used throughout the schema:

NO      = Number
CUST    = Customer
QTY     = Quantity

This database can be accessed by COBOL, QUERY, AQ and PROTOS.  Note
that the STREET-ADDRESS field is incompatible with QUERY, but AQ
can correctly add and modify the STREET-ADDRESS field.
>>
```

Naming of Items and Sets

Rule: *Names must be restricted to 15 characters; the only special character allowed in names is the dash (-).* This ensures that the names are compatible with V/3000 and COBOL.

The percent sign (%) should be replaced with the abbreviation "-PCT", and the hash sign (#) should be replaced with the abbreviation "-NO".

Item Layout

The easiest layout to implement, maintain and understand is to declare everything in the database sorted in alphabetic order. The items in the database should begin with a $PAGE command to separate the items from the header comment. Each item appears sorted by its name, regardless of the item's type or function.

In many IMAGE applications, the schema also acts as the data dictionary. For this reason, it is very important that every part of the database design be completely documented in the schema. Document each item as it is declared. To make each item stand out, the following layout should be used:

```
CUST-NO,                Z10;    << The customer number is used as a
                                   key field in the M-CUSTOMER dataset.
                                   It is also the defining path in
                                   the D-ORDER-DETAIL dataset.
                                >>
```

The item name, its type, and the comment start in the same column for every item. Each part of the item definition will stand out, and because the item names are in sorted order, the applications programmer can easily find a particular item.

Dataset Layout

Every dataset declaration must be preceeded by a header comment that describes the use of the dataset and any special facts that the programmer should be aware of.

When accessing the dataset from a COBOL program, it will be necessary to have a COBOL record which corresponds to the dataset. In order to prevent confusion between two occurrences of the same item as a field in several datasets, a prefix will be assigned to each of the variables in the COBOL buffer declaration. This prefix is selected by the database designer and must appear on the same line as the name of the database. For example:

```
<<  The M-CUSTOMER dataset gathers all of the static information
    about each customer into one dataset.  A customer must exist
    in this dataset before any sales are permitted to the
    customer.  This dataset also provides the necessary path
    into the D-SALES dataset.
>>

    NAME:   M-CUSTOMER,              MANUAL (1/2);  <<PREFIX=MCS>>
```

The AUTOMATIC, MANUAL or DETAIL keyword must always appear in the same column. This makes reading the schema easier, and by searching the file for a string (by using \L"NAME:" in QEDIT) it is possible to produce a nice index of dataset names, their types, and their prefixes. The following example prints an index of the STORE dataset names:

```
:RUN QEDIT.PUB.ROBELLE
/LQ STORE00.DB "NAME:"
NAME:  M-CUSTOMER,       MANUAL (1/2);       <<PREFIX = MCS>>
NAME:  M-PRODUCT,        MANUAL (1/2);       <<PREFIX = MPR>>
NAME:  M-SUPPLIER,       MANUAL (1/2);       <<PREFIX = MSU>>
NAME:  D-INVENTORY,      DETAIL (1/2);       <<PREFIX = DIN>>
NAME:  D-SALES,          DETAIL (1/2);       <<PREFIX = DSA>>
```

Rule: *Automatic master datasets have names that start with "A-".*

They must be declared immediately after the item declarations, separated from item declarations by a $PAGE command, and they must appear in alphabetic order.

Rule: *Manual master datasets have names that start with "M-".*

The manual master datasets follow the automatic master datasets, again preceded by a $PAGE command. Like the automatic masters, the manual master datasets must be declared in alphabetic sequence.

Rule: *Detail dataset names start with "D-".*

The detail datasets follow the manual master datasets, and the two are separated by a $PAGE command. The detail datasets also appear in alphabetic order.

Field Layout

Without exception, the fields in every dataset must be declared sorted alphabetically. There is a strong tendency to try to declare the fields within a dataset in some other type of logical grouping. Because this logical grouping exists only in the mind of the database designer and cannot be explicitly represented in IMAGE, it should never be used. By declaring fields in sorted order, the applications programmer can work much faster with the database, since no time has to be spent searching for fields within each dataset.

The database designer can still group fields together in a dataset by starting each field with the same prefix. If a dataset contains a group of costs, they might be called VAR-COSTS, FIX-COSTS and TOT-COSTS. To group these items together in the dataset, call them COSTS-VAR, COSTS-FIX and COSTS-TOT. This maintains the sorted field order in each dataset, while allowing for logical grouping of fields.

Most datasets contain one or more key fields. A key field is specified by following it with (). Because the () pair is sometimes hard to see, a comment should be included beside every key field, indicating that the field is a key. In a detail dataset, the primary key should include a comment to that effect. The following example shows how to declare the fields in a dataset:

```
<<  The D-SALES dataset gathers all of the sales records
    for each customer.  The primary on-line access is by customer,
    but it is necessary to have available the product sales
    records.  The PRODUCT-PRICE is the price at the time
    the product is ordered.  The SALES-TAX is computed based
    on the rate in effect on the DELIV-DATE.
>>

NAME:  D-SALES,          DETAIL (1/2);       <<PREFIX = DSA>>
```

```
ENTRY:
       CUST-ACCOUNT(!M-CUSTOMER)    <<KEY FIELD, PRIMARY PATH>>
      ,DELIV-DATE
      ,PRODUCT-NO(M-PRODUCT)        <<KEY FIELD>>
      ,PRODUCT-PRICE
      ,PURCH-DATE
      ,SALES-QTY
      ,SALES-TAX
      ,SALES-TOTAL
      ;
CAPACITY:  600;      <<D-SALES; 3 * CAP(M-CUSTOMER)>>
```

Capacities

Analysis of the data flow of the application should result in an approximate capacity for each dataset.

Rule: *The capacity of master datasets must be a prime number.*

To see if a number is prime :RUN the PRIME program contributed by Alfredo Rego. Master datasets should never be more than 80% full (see DBLOADNG below, under "Database Maintainence"), and detail datasets should never be more than 90% full.

The line with the capacity must be formatted in the following way:

```
CAPACITY:    211;      <<M-CUSTOMER,PRIME; ESTIMATED>>
```

The comment after the capacity gives a method for determining the approximate capacity of the dataset. Most detail datasets have a capacity that is related to the master datasets having paths into the detail datasets. These relationships should be described in the capacity comment.

By doing a \L"CAPACITY", it is possible to obtain quickly an index of the capacity of each dataset in the schema. Because the capacity is always the last line of each dataset declaration, doing a \L"M-CUSTOMER" will identify the beginning and ending declarations for the M-CUSTOMER dataset. The following example lists the capacity of the datasets in the STORE database:

```
:RUN QEDIT.PUB.ROBELLE
/LQ STORE00.DB "CAPACITY:"
CAPACITY:  211;      <<M-CUSTOMER,PRIME; ESTIMATED>>
CAPACITY:  307;      <<M-PRODUCT,PRIME; ESTIMATED>>
CAPACITY:  211;      <<M-SUPPLIER,PRIME; ESTIMATED>>
CAPACITY:  450;      <<D-INVENTORY; 2 * CAP(M-SUPPLIER)>>
CAPACITY:  600;      <<D-SALES; 3 * CAP(M-CUSTOMER)>>
```

Final Checkout

After the schema is entered into a file, it must be :RUN through the schema processor, and any typing mistakes should be eliminated:

```
:FILE DBSTEXT=STORE00.DB
:FILE DBSLIST;DEV=LP;CCTL
:RUN DBSCHEMA.PUB.SYS;PARM=3
```

The table produced at the end of the schema should be studied. The following anomalies should be checked:

1. Large-capacity master datasets with a blocking factor less than four (either increase the BLOCKMAX size to 1024, or change the master dataset to a detail dataset with an automatic master dataset).
2. The blocksize is too small (IMAGE optimizes the blocking factor to minimize disc space); use REBLOCK of ADAGER to increase the blocking factor. The blocksize of all dataset blocks should be as close to the BLOCKMAX size as possible.
3. Are there more than two paths into a detail dataset? If there are, can some of them be deleted?

Establishing the Programming Context

By using IMAGE, the COBOL programmer's job should be simplified, since all access to the database is done through the well-defined IMAGE procedures. Like most powerful tools, IMAGE (and COBOL) can be abused by the unsuspecting user.

Rule: *Define a standard IMAGE communication area*

and put this area in the COPYLIB.

The starting point for using IMAGE is the standard parameter area, which includes the IMAGE status area, the various access modes used, a variable for the database password, and a number of utility variables which are needed when using IMAGE. For example:

```
05  DB-ALL-LIST              PIC X(2) VALUE "@ ".
05  DB-SAME-LIST             PIC X(2) VALUE "* ".
05  DB-NULL-LIST             PIC S9(4) COMP VALUE 0.
05  DB-DUMMY-ARG             PIC S9(4).
05  DB-PASSWORD              PIC X(8).
05  DB-MODE1                 PIC S9(4) COMP VALUE 1.
05  DB-KEYED-READ            PIC S9(4) COMP VALUE 7.
05  DB-STATUS-AREA.
    10  DB-COND-WORD         PIC S9(4) COMP.
        88  DB-STAT-OK           VALUE ZEROS.
        88  DB-END-OF-CHAIN      VALUE 15.
        88  DB-BEGIN-OF-CHAIN    VALUE 14.
        88  DB-NO-ENTRY          VALUE 17.
        88  DB-END-FILE          VALUE 11.
        88  DB-BEGIN-FILE        VALUE 10.
    10  DB-STAT2             PIC S9(4) COMP.
    10  DB-STAT3-4           PIC S9(9) COMP.
    10  DB-CHAIN-LENGTH      PIC S9(9) COMP.
        88  DB-EMPTY-CHAIN   VALUE ZEROS.
    10  DB-STAT7-8           PIC S9(9) COMP.
    10  DB-STAT9-10          PIC S9(9) COMP.
```

Rule: *Establish naming standards for all variables associated with IMAGE databases.*

Standard prefixes must be used on all database variables, including the database, dataset, data field and dataset buffer declarations. A suggestion is to start all database variables with "DB-", all dataset names with "DB-SET-", and all database buffer declarations with "DB-BUFFER-". Data field names are prefixed by the special dataset prefix (which the designer established in the schema), so that each field has a unique name. For example:

```
01  DATASET-M-PRODUCT.

    05  DB-SET-M-PRODUCT             PIC X(10) VALUE "M-PRODUCT;".

    05  DB-BUFFER-M-PRODUCT.
        10  MPR-PRODUCT-DESC         PIC X(20).
        10  MPR-PRODUCT-NO           PIC 9(8).
```

Field Lists

The selection of the type of field lists depends on the answer to this question: Can your total application be recompiled in a weekend?

Rule: *Use "@" field list is you can recompile in a weekend (prepare a COPYLIB member for each dataset); use "*" field list otherwise and hire a DBA!*

If the answer to the question is "yes," the at ("@") field list and full buffer declarations should be used when accessing the database. This method requires that all dataset buffers be declared and added to the COPYLIB. If a dataset changes, the buffer declaration must be changed in the COPYLIB, and all affected programs must be recompiled. The simplest solution is to recompile the complete application system whenever a dataset changes.

There must be two complete COPYLIBs available for every application. One is for production, and one is for development.

Rule: *Use a test COPYLIB during development. Double-check that all existing programs will recompile and :RUN correctly before moving the new COPYLIB into production!*

When a database is restructured, the buffer declarations are first changed in the development COPYLIB. When the new database is put into production, the development COPYLIB is also moved into production, as well as any programs that required modification or recompilation.

If the application system is so large that it cannot be recompiled in a weekend, it should use partial field lists and the same ("*") field list. This requires that an application program declare a matching field list and buffer

area for each dataset that it accesses. The field list declares the minimum subset of the dataset that the application program needs.

Because partial field lists are more expensive at run time, the applications programmer must code a one-time call to DBGET for every dataset that the application program will use. The same ("*") field list is used on all subsequent DBGET calls. Note that this can cause problems if a common subroutine is called that uses one of the same datasets, but with a different field list.

In order to maintain an application with partial field lists, there must be a way to cross reference every program/dataset relationship. When a dataset changes, the cross reference system is checked to see which programs use the dataset. Each of these programs must be examined to see if it is affected by the change to the dataset. It is not enough to fix the COPYLIB and recompile, since the field declarations are in the individual source files, not in the COPYLIB file.

Dataset Buffers

The database designer assigns a short, unique prefix to each dataset of each database. These prefixes are used in the declaration of the database buffers for the datasets. In addition, dataset buffer declarations must include all 88-level definitions for flags, and sub-definitions for IMAGE fields that are logically sub-divided within the application.

The following is the full buffer declaration for the M-CUSTOMER dataset of the STORE database. Note that each variable is prefixed with "MCS-", which is the prefix that was assigned by the database designer.

```
01  DB-BUFFER-M-CUSTOMER.
    05  MCS-CITY                    PIC X(12).
    05  MCS-CREDIT-RATING           PIC S9(4)V9(5) COMP.
    05  MCS-CUST-ACCOUNT            PIC 9(10).
    05  MCS-CUST-STATUS             PIC X(2).
        88  MCS-CUST-ADVANCE        VALUE "10".
        88  MCS-CUST-CURRENT        VALUE "20".
        88  MCS-CUST-ARREARS        VALUE "30".
        88  MCS-CUST-INACTIVE       VALUE "40".
    05  MCS-NAME-FIRST              PIC X(10).
    05  MCS-NAME-LAST               PIC X(16).
    05  MCS-STATE-CODE              PIC X(2).
    05  MCS-STREET-ADDRESS          PIC X(25) OCCURS 2.
    05  MCS-ZIP-CODE.
        10  MCS-ZIP-CODE-1          PIC X(3).
        10  MCS-ZIP-CODE-2          PIC X(3).
```

Repeated items should be declared with an occurs clause, or sub-divided, whichever the application requires. For example, a cost field may be declared as a repeated item representing fixed, variable, overhead, and labor costs. Rather than declare the costs field as a repeated item in the actual buffer declaration, subdivide it into the four costs. For example, assume a declaration for costs such as:

```
COSTS,          4J2;    <<Cost of an item.  Each cost has two
                          decimal points and the cost item
                          is broken down as follows:
                          COSTS(1) = Variable costs
                          COSTS(2) = Fixed costs
                          COSTS(3) = Overhead costs
                          COSTS(4) = Labour costs
                        >>
```

Assuming that the COSTS field was declared in the D-INVENTORY dataset, which has a prefix of "DIN", the following buffer declaration would be used for the COSTS field:

```
01  DB-BUFFER-D-INVENTORY.
    05  DIN-COSTS.
        10  DIN-VARIABLE-COSTS      PIC S9(7)V9(2) COMP.
        10  DIN-FIXED-COSTS         PIC S9(7)V9(2) COMP.
```



```
        10  DIN-OVERHEAD-COSTS      PIC S9(7)V9(2) COMP.
        10  DIN-LABOUR-COSTS        PIC S9(7)V9(2) COMP.
```

Rule: *Prepare sample COBOL calls to IMAGE in source files, with one IMAGE call per file.*

The sample IMAGE calls should be organized with one parameter per line. When programming, these template IMAGE calls must be copied into the COBOL program and modified with the database name, dataset name, and any other necessary parameters.

General purpose SECTIONS, declared in the COPYLIB, should NOT be used for the IMAGE calls. These SECTIONS obscure the meaning of the COBOL code. In addition, they can cause unnecessary branches across segment boundaries.

A scheme for handling fatal IMAGE errors must be declared, and the sample IMPAGE calls should refer to the fatal-error section. Here is a sample call to the IMAGE routine DBFIND:

```
CALL "DBFIND" USING DB-
                    DB-SET-
                    DB-MODE1
                    DB-STATUS-AREA
                    DB-KEY-
                    DB-ARG-
 IF NOT DB-STAT-OK AND NOT DB-NO-ENTRY THEN
     PERFORM 99-FATAL-ERROR.
```

The fatal-error section (99-FATAL-ERROR) should call DBEXPLAIN. It should also cause the program to abort, and the system job-control word should be set to a fatal state. Note that just using STOP RUN will not set the system job-control word to a fatal state. The following is an example of a fatal-error section. The routine MISQUIT calls the QUIT intrinsic, which causes the program to abort.

```
$PAGE "[99]  FATAL ERROR"
*****************************************************************
*  THIS SECTION DOES THE FOLLOWING:                             *
*  1.  CALLS DBEXPLAIN WITH THE IMAGE STATUS AREA.              *
*  2.  CALLS MISQUIT TO ABORT THE PROGRAM.                      *
*                                                               *
*  NOTE:  THIS MODULE MUST ONLY BE CALLED AFTER A FATAL ERROR*
*         HAS OCCURED WHEN CALLING AN IMAGE ROUTINE.            *
*                                                               *
*****************************************************************

 99-FATAL-ERROR                  SECTION.

     CALL "DBEXPLAIN" USING DB-STATUS-AREA.

     CALL "MISQUIT" USING DB-COND-WORD.

 99-FATAL-ERROR-EXIT.  EXIT.
```

Rule: *Avoid tricky data structures,* especially if they cannot be easily retrieved and displayed with the available tools (QUERY, AQ, PROTOS, QUIZ, etc.).

Some examples of data structures to avoid: (1) julian dates; (2) bit maps; (3) alternate record structures (REDEFINES); (4) implied and composite keys/paths; and (5) implied description structures. The more complicated the database structure, the more likely it is that programming or system errors will be created as a result of the database design.

Database Maintenance

There are a number of steps that the database administrator must take in order to guarantee that a database remains clean after it is implemented. A number of standard programs must be run against each production database at least once a month; others must be run daily.

Backup

A number of other people have commented on the backup problem of databases,[12] but the problem is important enough to deserve comment again. Most HP3000 shops do a full backup once a week and a partial backup once a day. This is normally sufficient for most purposes (e.g., source files, PUB.SYS, utilities), but it is not adequate for most IMAGE applications. An IMAGE database consists of several interrelated files. A database that is missing one dataset is nearly useless.

Rule: *EVERY backup tape should include ALL of the*

files of ALL of the database that are used in day-to-day applications.

There should be an easy way to store complete databases onto partial backup tapes, without having to do selective stores. The BACKUP program (available from the San Antonio Swap Tape) helps solve this problem. The BACKUP program is run once a day against every production database. It accepts the database name as input and causes the last-modified date to be changed to today's date on every file of the database. This causes the entire database to be included on the daily partial backup.

In addition, the BACKUP program prints a listing with the following information: the dataset name, the current number of entries in the dataset, and the capacity of the dataset. Further, the BACKUP program examines the relationship between the number of entries and the capacity of each dataset, and prints a warning if it thinks the capacity is too small. This listing must be checked daily, in order to have time to expand the capacity of a dataset before it is exceeded.

Measuring Database Performance (DBLOADNG)

The performance of a given database will change as the database matures.

Rule: *The performance of every application database should be measured at least once a month.*

There is one program that will measure, in great detail, the performance of an IMAGE database. This program is DBLOADNG,[1-12] and it is available from the HPIUG contributed library.

DBLOADNG examines the performance of both master and detail datasets, and reports a large number of statistics. The most important are the percentage of secondaries in master datasets, and the elongation of detail datasets.

If there are a large number of secondaries in master datasets, either the hashing algorithm is not working well, or the capacity of the dataset needs to be increased. Note that the hashing performance of a key, such as customer number, can be improved by adding a check digit to every customer number.

The "elongation" of a detail dataset indicates whether logically related records are being stored physically adjacent. For primary paths, the elongation factor should be very small (1=perfect), since IMAGE tries to place records of a primary-path in the same disc block (see the DBLOADNG documentation and *Optimizing IMAGE: An Introduction.*[1]

If the performance of detail datasets is very poor because logically related records have been spread around the disc, there is only one solution: RELOAD the database using DBUNLOAD/DBLOAD. This will cause the detail dataset to be organized along the primary path, and could result in significant performance improvements.

Logical Database Maintenance

During the design phase of an IMAGE database, many logical assumptions are made about the data in the database. Some assumptions might be: (1) status fields, which are two characters long in a detail dataset, but have a long description in a master dataset; (2) keys that are stored in detail datasets, but do not have an explicit path into a master dataset; and (3) IMAGE chains that are limited to a specific length (e.g., one address per customer) or a range of lengths (e.g., no more than 10 items per order).

Rule: *When designing a database, keep a list of logical assumptions.*

These assumptions are dangerous, because they must be maintained by the application software, not by IMAGE.

Rule: *A program to check logical assumptions should be implemented for every application system.*

This program is often called DBREPORT, and its purpose is to check these logical assumptions. DBREPORT is often left until last, and often never implemented. This is unfortunate, since the DBREPORT program is *the* most important program in an application system.

In Alfredo Rego's paper, *DATABASE THERAPY: A practitioner's experiences*[12], he describes periodic checkups for a database. The following is taken from his paper:

> Please notice that a good diagnosis system must be nasty and sadistic by nature. It has as its primary objective to FIND ERRORS, not to certify a system as being error-free (there is no such system anyway!). A good diagnosis system must also be extremely patient and humble, since it will fail many times. Please keep in mind that there is a psychological inversion in effect here: A good diagnosis system fails if it does not detect any errors. And most of the time it will not detect any errors, since we hope and assume that the entity being tested is reasonably error-free."[12]

The DBREPORT program must be designed with Alfredo's philosophy in mind. It should check EVERY dataset in an application, and it should check EVERY record for logical consistency. This includes simple checks to see that every field in every dataset is within a reasonable limit. Examples of this are status fields that take on values from 1 to 10, but which are implemented as J1. A J1 variable can take on values from −32768 to +32767, which is certainly a larger range than 1 to 10.

The DBREPORT program must check all logical dataset relationships. What happens if every customer record has its address in a detail dataset? If the system crashes while the user is adding a new customer, the address record may not be added. DBREPORT must

check for these types of relationships (what will your billing program do when it can't find an address?).

ADAGER

Rule: *If an application system is going to depend on IMAGE, ADAGER is a requirement, not an option.*

ADAGER provides all of the restructuring facilities necessary to maintain IMAGE databases; these transformations cannot be accoplished with DBLOAD/DBUNLOAD. Without ADAGER, numerous conversion programs must be written.

While DBLOAD/DBUNLOAD can be used for some simple database restructuring, it is prone to err. ADAGER is designed to be friendly to the end user, but, more importantly, ADAGER guides the user through every phase of the database restructuring process.

ADAGER provides a powerful facility, but it can also be misused by the unsuspecting. In order to make ADAGER changes effectively, test them first on a development database. Following changes to the database structure, the application programs must be recompiled (with buffers changed in the development COPYLIB), and each program must be tested against the new database.

Currently, ADAGER cannot be run from batch (at least, not conveniently), nor does it produce a hard-copy audit trail of the changes to a database.

Rule: *ADAGER must be run on a printing terminal.*

Keep the listing of the ADAGER changes to the test database. Use it to verify that the changes to the production database match exactly the changes to the test database. After changing the production database, move the development COPYLIB into production and recompile all affected programs. File the hard-copy listing of the ADAGER changes and keep it for future reference.

Because the schema is also used as the data dictionary, it must be modified to indicate the new database design. ADAGER's SCHEMA function can be used to double check that all schema changes were made properly. When modifying the database schema, be sure to apply all of the rules in the Schema section of this paper.

BIBLIOGRAPHY

To gain a complete understanding of IMAGE, study the references in this bibliography. A suggested order of study is: References 6, 7, 9, 10 and 11 for more ideas on database design; 5 for some hints on common programming errors; and 1, 3, 8, 12 and 13 for notes on optimizing IMAGE databases and application systems in general. Reference 1 is an excellent introduction to database optimization, and it includes a discussion of the DBLOADNG program.

[1]Rick Bergquist, *Optimizing IMAGE: An Introduction,* HPGSUG 1980 San Jose Proceedings.

[2]Gerald W. Davidson, *Image Locking and Application Design,* Journal of the HPGSUG, Vol. IV, No. 1.

[3]Robert M. Green, *Optimizing On-Line Programs,* Technical Report, second edition, Robelle Consulting Ltd.

[4]Robert M. Green, *SPLAIDS2 Software Package,* contains date editing routine (SUPRDATE) available from Robelle Consulting Ltd.

[5]Robert M. Green, *Common Programming Errors With IMAGE/3000,* Journal of the HPGSUG, Vol. I, No. 4.

[6]Hewlett-Packard, IMAGE/3000 Reference Manual.

[7]Karl H. Kiefer, *Data Base Design – Polishing Your Image,* HPGSUG 1981 Orlando Proceedings.

[8]Jim Kramer, *Saving the Precious Resource – Disc Accesses,* HPGSUG 1981 Orlando Proceedings.

[9]Ken Lessey, *On Line System Design and Development,* HPGSUG 1981 Orlando Proceedings.

[10]Brian Mullen, *Hiding Data Structures in Program Modules,* HPGSUG 1980 San Jose Proceedings.

[11]Alfredo Rego, *Design and Maintenance Criteria for IMAGE/3000,* Journal of the HPGSUG, Vol. III, No. 4.

[12]Alfredo Rego, *DATABASE THERAPY: A practitioner's experiences,* HPGSUG 1981 Orlando Proceedings.

[13]Bernadette Reiter, *Performance Optimization for IMAGE,* HPGSUG 1980 San Jose Proceedings.

Using COBOL, VIEW and IMAGE A Practical Structured Interface for the Programmer

Peter Somers
Cape Data, Inc.

INTRODUCTION

VIEW or V/3000, Hewlett-Packard's screen handler offers a convenient and versatile method of data collection. To fully utilize the capabilities of VIEW requires the application programmer to go beyond the routines available using the ENTRY program. Ideally the data entry routine will include complete editing including IMAGE data base checking and comprehensive error messages. The routine should allow the programmer to quickly "plug in" new applications and easily perform maintenance. Additionally the program will provide utility routines for data confirmation, screen refreshing, paging, etc.

At our shop, Cape Data, we developed a general purpose VIEW and IMAGE interface program. This program written in structured COBOL allows new applications to go up, with custom editing, in a fraction of the time previously required. The following discussion will cover this interface routine and its application. I will assume that the user has basic knowledge of both VIEW and COBOL.

TABLE OF CONTENTS

1. Screen Design

When designing your VIEW input screens using FORMSPEC, the following techniques will help you get the most out of VIEW.

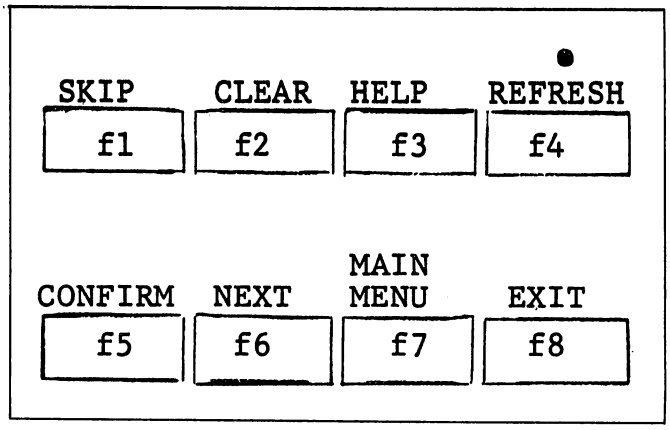
A. *Error Message Fields:* Add error message fields during form design wherever needed. Place the error message field next to or under the corresponding data field. The error message fields will remain invisible unless your program writes a message to the field. Create the field with an enhancement of B (blink) and a field type of D (display) and an initial value of spaces (Fig. 1, Field 3). The last 24th line of the screen is reserved for program error messages.

B. *VIEW Editing:* As a general rule let VIEW do as much editing as possible. On numeric fields let VIEW zero fill and test for numeric input during the FIELD portion of VIEW's editing. On alpha-numeric input fields allow VIEW to left justify the data and optionally upshift lower case characters (Fig. 1, Field 2).

C. *Title:* We reserve a Title area on all forms using Field #1. The title field is initialized by VIEW to a save field value. The VIEW forms file can contain the title and any other constants in SAVE FIELDS.

Function Keys

When using a formatted screen program with a Hewlett-Packard 2640-2645 type terminal, a special set of function keys are used by the programs. The 8 function keys are located on the upper right hand side of the terminal's keyboard. They are labeled with blue letters *f*1 through *f*8 in 2 rows. A blank Hewlett-Packard template labels the function keys (#7120-5525). On the 2620 family of terminals, the keys are labeled programmatically.



Function Keys

The function keys are used as follows in the interface program: *f1* SKIP — This key will cause the cursor to

skip to the next block of data. This is useful if you have a number of fields to skip. The tab key only skips a field at a time where the SKIP key will skip to the next block of data.

f2 CLEAR (RESET) — This key causes the screen to clear all fields and set the initial field values (usually blanks). This key is useful if you have created a mess on the screen and want to start over again.

f3 HELP — This key will cause the program to display an instruction screen. A special set of instructions can be displayed relating to the particular form on the screen when the HELP key was pushed. When you have finished reading the HELP instructions, push the ENTER key to return to the last form or the MENU (f7) KEY to return to the MAIN MENU.

f4 REFRESH — This key resets the terminal, erases the screen and brings up a fresh copy of the form. If you loose your form due to a power or line failure or the terminal hangs up, the REFRESH KEY will restore the terminal to normal operation.

f5 CONFIRM — This key is used when you have changed a record in the EDIT mode, or if the program wants confirmation that the data on the screen is acceptable. The program will prompt with a message at the bottom of the screen when a CONFIRM is desired. Before a CONFIRM is requested the data must pass all normal program edits.

Note:

The terminal normally does not read data when the function keys are pushed. If you need to read the screen contents after a function key has been pushed call the IMMVREADFIELD Subroutine to force a read.

f6 NEXT — This key will cause the program to go to the NEXT form or next step.

f7 MAIN MENU — This key causes the program to display the MAIN MENU SELECTION form. Use this key to change from one program mode to another.

f8 EXIT — This key ends the program.

3. COBOL Application Program

A. *General:* The attached COBOL program has sufficient structure to allow the programmer to readily plug in applications without having to spend additional time coding VIEW procedures. At Cape Data, I have used this program layout to do extensive data entry routines which edit against the IMAGE data base, and provide detailed error messages and help routines. The program procedure division consists of 3 parts:

1. Opening
2. Main Loop
3. Closing

The program contains routines which call the VIEW procedures listed in Fig. 7. The program also contains special VIEW data fields in working storage.

1. Opening — The program opens the terminal, forms file, the data base and displays the MENU screen.

2. Main Loop — After opening the program performs the Main Loop until either the *f8* (exit) function key is pushed, or the program encounters a fatal error. The Main Loop consists of 3 parts:

   a. Read Keys and Screen
   b. Edit Input
   c. Process Input (if valid) and Refresh Screen

3. Closing — The program closes the terminal file, forms file and data base.

B. *Program Data Division Considerations:* The working storage area contains the buffers needed by the various VIEW procedures used. Every VIEW procedure called uses the VIEW-COM buffer (Fig. 2). Most of the fields useful to the programmer have self-explanatory names. Remember the V-Language field must be set to zero for a COBOL program.

The data area passed between the program and VIEW (using VGETBUFFER and VPUTBUFFER calls) is defined as DATA-BUF (Fig. 3). This Buffer is redefined for each screen layout. Note the title field and error message fields.

The program uses a forms table which contains the MENU selection character, form number, next form number, and help form number for each routine. When the user makes a selection from the Screen Menu, the program scans this table to find the corresponding screen references (Fig. 4). The program picks up the Form Name corresponding to the Screen Number from the Form Name Table (Fig. 6). Additional routines and forms can quickly be added by making additional entries in the tables.

3. Program Main Loop

The program goes to the MAIN LOOP and remains there until the user pushes the *f8* (exit) function key. The program performs a terminal read (VREADFIELDS) each time either a function key or the enter key is depressed. The program then tests to see if any function keys were pushed (VIEW returns the key number pushed into the last key field of the VIEW-COM buffer).

If the ENTER key was pushed (key zero) the program will perform the edit routine corresponding to the screen routine selected. Within each edit routine the program does the following steps (Fig. 5):

1. Zeros the field error array and set the field count.
2. Performs VIEW edits (VFIELDEDITS).
3. Get the data buffer from VIEW (VGETBUFFER)
4. Clear the error message fields.
5. Performs any user defined edits (if an error is detected, a flag is set in the field error array and a message is moved to the appropriate error field)
6. The data area is sent back to VIEW (VPUTBUFFER)
7. Perform VIEW edits again (VFIELDEDITS). This zero fills and justifies data.

8. The field error array is scanned and any error fields are set to blink (VSETERROR).
9. The screen is updated and displayed (VSHOWFORM).

If the routine passes all edits, the program then goes to the corresponding valid record routine. If an error exists or a function key was depressed, the program will do the appropriate error and screen enhancing routines.

4. SPL Forced Read Subroutine

```
$CONTROL SUBPROGRAM
<< KEPT AS IMMREAD >>
<< THIS ROUTINE IS CALLED TO FORCE          >>
<<  AN IMMEDIATE READ (RE-READ FOR DATA)    >>
<<  IN PARTICULAR CASES WHERE THE USER      >>
<<  USED A SOFT KEY TO INDICATE ACTIONS     >>
<<  HE/SHE WANTS PERFORMED WITH THE DATA     >>
<<  THAT HAS BEEN ENTERED ON THE SCREEN      >>
<< SINCE THE HITTING OF A SOFT KEY DOES     >>
<<  NOT TRANSFER THE ACTUAL BUFFER DATA      >>
<<  A CALL TO THIS ROUTINE OR ONE LIKE       >>
<<  IT IS NEEDED TO GET THE SCREEN DATA      >>
<<  INTO THE PROGRAM WHERE IT CAN BE         >>
<<  WORKED UPON                              >>
<< IN COBOL, THE CALL WOULD BE              >>
<<   CALL "IMMVREADFIELDS" USING VCONT      >>
<<   WITH VCONT BEING THE V/3000 CONTROL    >>
<<   AREA                                   >>
<< IN THE USER PROGRAM, THIS SHOULD BE      >>
<<  TREATED EXACTLY AS IF IT HAD BEEN A     >>
<<  CALL TO "VREADFIELDS"                   >>
BEGIN PROCEDURE IMMVREADFIELDS(CONT);
INTEGER ARRAY CONT;
BEGIN
PROCEDURE VREADFIELDS(C);
INTEGER ARRAY C;
OPTION EXTERNAL;
CONT(55).(13:2):=%1;
VREADFIELDS(CONT);
CONT(55).(13:2):=0;
END;
END.
```

```
FORMSPEC  VERSION A.00.01
FORMS FILE: BUDGFORM.DEVELOP.FDEVELOP

FORM: VENDOR_DATA
   REPEAT OPTION: N

   NEXT FORM OPTION: C
   NEXT FORM: BUDMAINT_HELP

VENDOR MASTER SPECIFICATONS, INPUT & EDITING
********* ********* ********* ********* ********* ********* ********* **********
          TITLE_____________________________________________________
              V E N D O R - M A S T E R   I N F O R M A T I O N

              VENDOR NUMBER   V_NBR_        VEND_ERR_______________

              VENDOR'S NAME   VEND_NAME_____________________
              VENDOR ADDRESS  VEND_ADDR_____________________
                              VEND_ADDR2____________________
              VENDOR'S CITY   VEND_CITY_________ STATE: ST  ZIP: V_ZIP_____


              PAYMENT ADDRESS P_ADDRESS_____________________
                              P_ADDRESS2____________________
              PAYMENT CITY    P_CITY____________ STATE: PS  ZIP: P_ZIP_____

     (PAYMENT ADDRESS USED ONLY IF YOU WANT PAYMENTS GO TO A DIFFERERNT ADDRESS)

                   VENDOR'S PHONE NUMBER: (AC_) EXC-PHON

           1099 CODE IN        VENDOR CODE VC          VENDOR STATUS VS
                       CODE_ERR_____________   STATUS_ERR_____________
********* ********* ********* ********* ********* ********* ********* **********


FIELD: TITLE
   NUM: 1    LEN: 69    NAME: TITLE          ENH: NONE  FTYPE: O  DTYPE: CHAR
   INIT VALUE:
*** PROCESSING SPECIFICATIONS ***
INIT
SET TO SFTITLE


FIELD: V_NBR
   NUM: 2    LEN: 6     NAME: V_NBR          ENH: 1     FTYPE: R  DTYPE: DIG
   INIT VALUE:
*** PROCESSING SPECIFICATIONS ***
FIELD
JUSTIFY RIGHT
FILL LEADING "0"


FIELD: VEND_ERR
   NUM: 3    LEN: 26    NAME: VEND_ERR       ENH: B     FTYPE: O  DTYPE: CHAR
   INIT VALUE:


FIELD: VEND_NAME
   NUM: 4    LEN: 30    NAME: VEND_NAME      ENH: 1     FTYPE: R  DTYPE: CHAR
   INIT VALUE:
```

Figure 1

```
6.4
6.5     01 VIEW-COM.
6.6          05 V-STATUS                 PIC S9(4)   COMP   VALUE ZERO.
6.7          05 V-LANGUAGE               PIC S9(4)   COMP   VALUE ZERO.
6.8          05 V-COM-AREA-LEN           PIC S9(4)   COMP   VALUE 60.
6.9          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
7            05 VIEW-MODE                PIC S9(4)   COMP   VALUE ZERO.
7.1          05 LAST-KEY                 PIC S9(4)   COMP   VALUE ZERO.
7.2          05 V-NUM-ERRS               PIC S9(4)   COMP   VALUE ZERO.
7.3          05 V-WINDOW-ENH             PIC S9(4)   COMP   VALUE ZERO.
7.4          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
7.5          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
7.6          05 V-CFNAME                 PIC X(15)   VALUE SPACES.
7.7          05 FILLER                   PIC X       VALUE SPACES.
7.8          05 V-NFNAME                 PIC X(15)   VALUE SPACES.
7.9          05 FILLER                   PIC X       VALUE SPACES.
8            05 V-REPEAT-OPT             PIC S9(4)   COMP   VALUE ZERO.
8.1          05 V-NF-OPT                 PIC S9(4)   COMP   VALUE ZERO.
8.2          05 V-NBR-LINES              PIC S9(4)   COMP   VALUE ZERO.
8.3          05 V-OBUF-LEN               PIC S9(4)   COMP   VALUE ZERO.
8.4          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
8.5          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
8.6          05 V-DELETE-FLAG            PIC S9(4)   COMP   VALUE ZERO.
8.7          05 V-SHOW-CONTROL           PIC S9(4)   COMP   VALUE ZERO.
8.8          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
8.9          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9            05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.1          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.2          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.3          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.4          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.5          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.6          05 V-NUM-RECS               PIC S9(6)   COMP   VALUE ZERO.
9.7          05 V-REC-NBR                PIC S9(6)   COMP   VALUE ZERO.
9.8          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
9.9          05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10           05 V-TERM-FILE-NBR          PIC S9(4)   COMP   VALUE ZERO.
10.1         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.2         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.3         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.4         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.5         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.6         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.7         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.8         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
10.9         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
11           05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
11.1         05 FILLER                   PIC S9(4)   COMP   VALUE ZERO.
11.2
```

Figure 2

```
13.3     01  DATA-BUF.
13.4         05 FILLER          PIC X(69).
13.5         05 DATA-IN         PIC X(443).
13.6
13.7     01 MENU-DATA       REDEFINES DATA-BUF.
13.8         05 FILLER                 PIC X(69).
13.9         05 SELECT-IN              PIC X.
14           05 SELECT-ERR           PIC X(26).
14.1
14.2     01 VEND-IN       REDEFINES DATA-BUF.
14.3         05 FILLER                 PIC X(69).
14.4         05 VEND-NBR               PIC X(6).
14.5         05 VEND-NBR-ERR           PIC X(26).
14.6         05 VEND-NAME              PIC X(30).
14.7         05 S-ADDRESS              PIC X(30).
14.8         05 S-ADDRESS2             PIC X(30).
14.9         05 S-CITY                 PIC X(20).
15           05 S-STATE                PIC X(2).
15.1         05 S-ZIP                  PIC X(10).
15.2         05 P-ADDRESS              PIC X(30).
15.3         05 P-ADDRESS2             PIC X(30).
15.4         05 P-CITY                 PIC X(20).
15.5         05 P-STATE                PIC X(2).
15.6         05 P-ZIP                  PIC X(10).
15.7         05 PHONE-NBR.
15.8          10 PHONE-AC              PIC X(3).
15.9          10 PHONE-EX              PIC X(3).
16            10 PHONE-NO              PIC X(4).
16.1         05 FLAG-1099              PIC X(2).
16.2         05 VEND-CODE              PIC X(2).
16.3         05 VENDOR-STATUS          PIC X(2).
16.4         05 VEND-CODE-ERR          PIC X(26).
16.5         05 VEND-STATUS-ERR        PIC X(26).
16.6
16.7     01 BUDG-IN                  REDEFINES DATA-BUF.
16.8         05 FILLER                 PIC X(69).
16.9         05 ACCT-NBR               PIC X(20).
17           05 BUDG-NBR-ERR           PIC X(26).
17.1         05 BUDG-NY-AMT            PIC X(13).
17.2
```

Figure 3

```
19      01 FORM-TABLE.
19.1        05 FORM-1.
19.2         10 FILLER                  PIC X     VALUE "Z".
19.3         10 FILLER                  PIC 9(4) COMP VALUE  1.
19.4         10 FILLER                  PIC 9(4) COMP VALUE  1.
19.5         10 FILLER                  PIC 9(4) COMP VALUE  14.
19.6        05 FORM-2.
19.7         10 FILLER                  PIC X     VALUE "A".
19.8         10 FILLER                  PIC 9(4) COMP VALUE  2.
19.9         10 FILLER                  PIC 9(4) COMP VALUE  1.
20           10 FILLER                  PIC 9(4) COMP VALUE  13.
20.1        05 FORM-3.
20.2         10 FILLER                  PIC X     VALUE "B".
20.3         10 FILLER                  PIC 9(4) COMP VALUE  3.
20.4         10 FILLER                  PIC 9(4) COMP VALUE  1.
20.5         10 FILLER                  PIC 9(4) COMP VALUE  13.
20.6        05 FORM-4.
20.7         10 FILLER                  PIC X     VALUE "C".
20.8         10 FILLER                  PIC 9(4) COMP VALUE  4.
20.9         10 FILLER                  PIC 9(4) COMP VALUE  1.
21           10 FILLER                  PIC 9(4) COMP VALUE  11.
21.1        05 FORM-5.
21.2         10 FILLER                  PIC X     VALUE "D".
21.3         10 FILLER                  PIC 9(4) COMP VALUE  2.
21.4         10 FILLER                  PIC 9(4) COMP VALUE  1.
21.5         10 FILLER                  PIC 9(4) COMP VALUE  13.
21.6        05 FORM-6.
21.7         10 FILLER                  PIC X     VALUE "E".
21.8         10 FILLER                  PIC 9(4) COMP VALUE  5.
21.9         10 FILLER                  PIC 9(4) COMP VALUE  1.
22           10 FILLER                  PIC 9(4) COMP VALUE  12.
22.1        05 FORM-7.
22.2         10 FILLER                  PIC X     VALUE "Z".
22.3         10 FILLER                  PIC 9(4) COMP VALUE  1.
22.4         10 FILLER                  PIC 9(4) COMP VALUE  1.
22.5         10 FILLER                  PIC 9(4) COMP VALUE  12.
22.6        05 FORM-8.
22.7         10 FILLER                  PIC X     VALUE "Z".
22.8         10 FILLER                  PIC 9(4) COMP VALUE  1.
22.9         10 FILLER                  PIC 9(4) COMP VALUE  1.
23           10 FILLER                  PIC 9(4) COMP VALUE  14.
23.1        05 FORM-9.
23.2         10 FILLER                  PIC X     VALUE "Z".
23.3         10 FILLER                  PIC 9(4) COMP VALUE  1.
23.4         10 FILLER                  PIC 9(4) COMP VALUE  1.
23.5         10 FILLER                  PIC 9(4) COMP VALUE  14.
23.6        05 FORM-10.
23.7         10 FILLER                  PIC X     VALUE "Z".
23.8         10 FILLER                  PIC 9(4) COMP VALUE  1.
23.9         10 FILLER                  PIC 9(4) COMP VALUE  1.
24           10 FILLER                  PIC 9(4) COMP VALUE  14.
24.1
24.2    01 FORM-SPEC-ARRAY          REDEFINES FORM-TABLE.
24.3        05 FORM-SPECS              OCCURS  10 TIMES.
24.4         10 FORM-ID                PIC X.
24.5         10 FORM-NBR               PIC 9(4)  COMP.
24.6         10 FORM-NEXT              PIC 9(4)  COMP.
24.7         10 FORM-HELP              PIC 9(4)  COMP.
```

Figure 4

```
24.9    01 FORM-NAME-TABLE.
25          05 FORM-1.
25.1         10 FILLER                PIC X(15) VALUE "BUDMAINT_MENU  ".
25.2         10 FILLER                PIC 9(4)  COMP VALUE  96.
25.3        05 FORM-2.
25.4         10 FILLER                PIC X(15) VALUE "VENDOR_DATA    ".
25.5         10 FILLER                PIC 9(4)  COMP VALUE 357.
25.6        05 FORM-3.
25.7         10 FILLER                PIC X(15) VALUE "BANK_MSTR_DATA ".
25.8         10 FILLER                PIC 9(4)  COMP VALUE 325.
25.9        05 FORM-4.
26           10 FILLER                PIC X(15) VALUE "BUDGET_LOAD    ".
26.1         10 FILLER                PIC 9(4)  COMP VALUE 128.
26.2        05 FORM-5.
26.3         10 FILLER                PIC X(15) VALUE "               ".
26.4         10 FILLER                PIC 9(4)  COMP VALUE 96.
26.5        05 FORM-6.
26.6         10 FILLER                PIC X(15) VALUE "               ".
26.7         10 FILLER                PIC 9(4)  COMP VALUE 96.
26.8        05 FORM-7.
26.9         10 FILLER                PIC X(15) VALUE "               ".
27           10 FILLER                PIC 9(4)  COMP VALUE  96.
27.1        05 FORM-8.
27.2         10 FILLER                PIC X(15) VALUE "               ".
27.3         10 FILLER                PIC 9(4)  COMP VALUE 96.
27.4        05 FORM-9.
27.5         10 FILLER                PIC X(15) VALUE "               ".
27.6         10 FILLER                PIC 9(4)  COMP VALUE 96.
27.7        05 FORM-10.
27.8         10 FILLER                PIC X(15) VALUE "               ".
27.9         10 FILLER                PIC 9(4)  COMP VALUE 96.
28          05 FORM-11.
28.1         10 FILLER                PIC X(15) VALUE "HELP_BANK_DATA ".
28.2         10 FILLER                PIC 9(4)  COMP VALUE 69.
28.3        05 FORM-12.
28.4         10 FILLER                PIC X(15) VALUE "HELP_BUDG_LOAD ".
28.5         10 FILLER                PIC 9(4)  COMP VALUE 69.
28.6        05 FORM-13.
28.7         10 FILLER                PIC X(15) VALUE "HELP_VENDOR    ".
28.8         10 FILLER                PIC 9(4)  COMP VALUE 69.
28.9        05 FORM-14.
29           10 FILLER                PIC X(15) VALUE "BUDMAINT_HELP  ".
29.1         10 FILLER                PIC 9(4)  COMP VALUE 69.
29.2        05 FORM-15.
29.3         10 FILLER                PIC X(15) VALUE "HELP_CREDIT_SUP".
29.4         10 FILLER                PIC 9(4)  COMP VALUE 69.
29.5
29.6    01 FORM-NAME-ARRAY   REDEFINES FORM-NAME-TABLE.
29.7        05 FORM-NAME-INFO       OCCURS  15 TIMES.
29.8         10 FORM-NAME               PIC X(15).
29.9         10 FORM-DATA-LEN           PIC 9(4)  COMP.
30
```

Figure 4a

```
48.3    102000-EDIT-A.
48.4        MOVE ZERO TO CHECK-RESULT.
48.5        MOVE ZERO TO FIELD-ZERO.
48.6        MOVE  16 TO FIELD-CNT.
48.7        PERFORM 805000-VIEW-EDIT.
48.8        PERFORM 807000-GET-BUFFER.
48.9        MOVE SPACES TO VEND-NBR-ERR, VEND-STATUS-ERR.
49          PERFORM 102100-CK-VEND-NBR.
49.1        PERFORM 102200-CK-VEND-CODE.
49.2        PERFORM 102300-CK-VEND-STATUS.
49.3        PERFORM 102400-CK-VEND-1099.
49.4
49.5        PERFORM 809000-PUT-BUFFER.
49.6        PERFORM 805000-VIEW-EDIT.
49.7        IF V-NUM-ERRS NOT = ZERO MOVE 1 TO CHECK-RESULT.
49.8        MOVE ZERO TO FIELD-LOC.
49.9        PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
50
50.1    102100-CK-VEND-NBR.
50.2        MOVE VEND-NBR OF VEND-IN TO ARGUMENT.
50.3        PERFORM 831000-GET-VEND-MSTR.
50.4        IF COND-WORD = 17 NEXT SENTENCE
50.5         ELSE MOVE "INVALID! DUPLICATE NUMBER" TO VEND-NBR-ERR
50.6          MOVE  1 TO FIELD-ERR (2)
50.7          MOVE  1 TO CHECK-RESULT.
50.8
50.9    102200-CK-VEND-CODE.
51          IF VEND-CODE OF VEND-IN = "VN" OR = "VM" OR = "DP"
51.1         NEXT SENTENCE
51.2          ELSE MOVE "INVALID VENDOR CODE!" TO VEND-CODE-ERR
51.3           MOVE  1 TO FIELD-ERR (19)
51.4           MOVE  1 TO CHECK-RESULT.
51.5
51.6    102300-CK-VEND-STATUS.
51.7        IF VENDOR-STATUS OF VEND-IN = "CR" OR = "XX"
51.8         NEXT SENTENCE
51.9          ELSE MOVE "INVALID STATUS CODE!" TO VEND-STATUS-ERR
52             MOVE  1 TO FIELD-ERR (20)
52.1           MOVE  1 TO CHECK-RESULT.
52.2
52.3    102400-CK-VEND-1099.
52.4        IF FLAG-1099 OF VEND-IN = SPACES OR = "Y "
52.5         NEXT SENTENCE
52.6          ELSE
52.7           MOVE  1 TO FIELD-ERR (18)
52.8           MOVE  1 TO CHECK-RESULT.
52.9
53
```

Figure 5

Summary of VIEW Procedures

PROCEDURE	FUNCTION
VCLOSEBATCH	Closes batch file.
VCLOSEFORMF	Closes forms file.
VCLOSETERM	Closes terminal file.
VERRMSG	Returns message associated with error code.
VFIELDEDITS	Edits field data and performs other field processing.
VFINISHFORM	Performs final processing specified for form.
VGETBUFFER	Reads contents of data buffer into user program.
VGETFIELD	Reads field from data buffer into user program.
VGETNEXTFORM	Reads next form into form definition area of memory; window and data buffer are not affected.
VGETtype	Reads field from data buffer to user program, converting data to specified type.
VINITFORM	Sets data buffer to initial values for form.
VOPENBATCH	Opens batch file for processing.
VOPENFORMF	Opens forms file for processing.
VOPENTERM	Opens terminal file for processing.
VPRINTFORM	Prints current form and data on offline list device.
VPUTBUFFER	Writes data from user program to data buffer.
VPUTFIELD	Writes data from user program to field in data buffer.
VPUTtype	Writes data of specified type from user program to data buffer, converting data to ASCII.
VPUTWINDOW	Writes message from user program to window area in memory for later display.
VREADBATCH	Reads record from batch file into data buffer.
VREADFIELDS	Reads input from terminal into data buffer.
VSETERROR	Sets error flag for data field in error; and moves error message to window area.
VSHOWFORM	Updates terminal screen, merging the current form, any data in buffer, and any message in window.
VWRITEBATCH	Writes data from data buffer to batch file.

Figure 6

COBOL VIEW Application Source Program

```
1
1.1   $CONTROL LIST,NOSOURCE,USLINIT,BOUNDS
1.2    IDENTIFICATION DIVISION.
1.3    PROGRAM-ID.
1.4        LAYOUT.
1.5   **THIS PROGRAM PROVIDES A LAYOUT FOR USING VIEW FORMS WITH COBOL.
1.6   ** THE PROGRAM DISPLAYS, EDITS, AND UPDATES MANY FORMS.
1.7   *** VERS 0.00      APRIL  9,1980.
1.8    AUTHOR.
1.9        P SOMERS.
2      INSTALLATION.
2.1        CAPE DATA INC.
2.2   *** (C)COPYRIGHT 1980 CAPE DATA INC. CAPE MAY,NEW JERSEY 08204
2.3    DATE-COMPILED.
2.4    ENVIRONMENT DIVISION.
2.5    CONFIGURATION SECTION.
2.6    SOURCE-COMPUTER. HP-3000.
2.7    OBJECT-COMPUTER. HP-3000.
2.8    INPUT-OUTPUT SECTION.
2.9
3      DATA DIVISION.
3.1
3.2    WORKING-STORAGE SECTION.
3.3    77  FBASE              PIC X(13) VALUE "  BUDGET.PUB;".
3.4    77  PASSWORD           PIC X(8)  VALUE "ABC1234;".
3.5    77  DSET-NAME          PIC X(16) VALUE SPACES.
3.6    77  NO-ITEM            PIC X(2)  VALUE "0;".
3.7    77  ITEM               PIC X(16) VALUE SPACES.
3.8    77  LIST               PIC X(30) VALUE SPACES.
3.9    77  ALL-ITEMS          PIC X(2)  VALUE "@;".
4      77  SAME-ITEMS         PIC X(2)  VALUE "*;".
4.1    77  ARGUMENT           PIC X(20) VALUE SPACES.
4.2    77  MODE1              PIC 9(4)  COMP  VALUE 1.
4.3    77  MODE2              PIC 9(4)  COMP  VALUE 2.
4.4    77  MODE3              PIC 9(4)  COMP  VALUE 3.
4.5    77  MODE5              PIC 9(4)  COMP  VALUE 5.
4.6    77  MODE7              PIC 9(4)  COMP  VALUE 7.
4.7    77  MODE-FLAG          PIC X.
4.8    77  LOC-FORM           PIC 9(4)  COMP.
4.9    77  LOC-FORM-NAME      PIC 9(4)  COMP.
5      77  LOC-FIND           PIC 9(4)  COMP.
5.1    77  FUTURE-FLAG        PIC 9.
5.2    77  NEW-FLAG           PIC 9.
5.3    77  LAST-RESULT        PIC 9 .
5.4    77  CHECK-RESULT       PIC S9(4)  COMP.
5.5    01  BELL               PIC X   VALUE " ".
5.6
5.7    01 STATUS-AREA.
5.8        05 COND-WORD       PIC S9(4)   COMP.
5.9        05 O-L             PIC S9(4)   COMP.
6          05 R-N             PIC S9(9)   COMP.
6.1        05 C-L             PIC S9(9)   COMP.
6.2        05 B-A             PIC S9(9)   COMP.
6.3        05 F-A             PIC S9(9)   COMP.
6.4
6.5    01 VIEW-COM.
6.6        05 V-STATUS              PIC S9(4)  COMP  VALUE ZERO.
```

```
6.7          05 V-LANGUAGE              PIC S9(4)  COMP  VALUE ZERO.
6.8          05 V-COM-AREA-LEN          PIC S9(4)  COMP  VALUE 60.
6.9          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
7            05 VIEW-MODE               PIC S9(4)  COMP  VALUE ZERO.
7.1          05 LAST-KEY                PIC S9(4)  COMP  VALUE ZERO.
7.2          05 V-NUM-ERRS              PIC S9(4)  COMP  VALUE ZERO.
7.3          05 V-WINDOW-ENH            PIC S9(4)  COMP  VALUE ZERO.
7.4          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
7.5          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
7.6          05 V-CFNAME                PIC X(15)  VALUE SPACES.
7.7          05 FILLER                  PIC X      VALUE SPACES.
7.8          05 V-NFNAME                PIC X(15)  VALUE SPACES.
7.9          05 FILLER                  PIC X      VALUE SPACES.
8            05 V-REPEAT-OPT            PIC S9(4)  COMP  VALUE ZERO.
8.1          05 V-NF-OPT                PIC S9(4)  COMP  VALUE ZERO.
8.2          05 V-NBR-LINES             PIC S9(4)  COMP  VALUE ZERO.
8.3          05 V-DBUF-LEN              PIC S9(4)  COMP  VALUE ZERO.
8.4          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
8.5          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
8.6          05 V-DELETE-FLAG           PIC S9(4)  COMP  VALUE ZERO.
8.7          05 V-SHOW-CONTROL          PIC S9(4)  COMP  VALUE ZERO.
8.8          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
8.9          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9            05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.1          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.2          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.3          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.4          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.5          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.6          05 V-NUM-RECS              PIC S9(6)  COMP  VALUE ZERO.
9.7          05 V-REC-NBR               PIC S9(6)  COMP  VALUE ZERO.
9.8          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
9.9          05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10           05 V-TERM-FILE-NBR         PIC S9(4)  COMP  VALUE ZERO.
10.1         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.2         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.3         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.4         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.5         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.6         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.7         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.8         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
10.9         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
11           05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
11.1         05 FILLER                  PIC S9(4)  COMP  VALUE ZERO.
11.2
11.3     01 V-FILE-NAME       PIC X(36).
11.4
11.5     01 ERR-MES-BUF             PIC X(76).
11.6     01 LEN-ERR-BUF             PIC S9(4)  COMP  VALUE 76.
11.7     01 LEN-ERR-MES             PIC S9(4)  COMP  VALUE ZERO.
11.8     01 TERM-FILE               PIC X(8)   VALUE SPACES.
11.9     01 DATA-LEN                PIC S9(4)  COMP .
12       01 FIELD-NBR               PIC S9(4)  COMP VALUE ZERO.
12.1     01 ACT-FIELD-LEN           PIC S9(4)  COMP VALUE ZERO.
12.2     01 NEXT-FIELD-NBR          PIC S9(4)  COMP VALUE ZERO.
12.3     01 NO-MESSAGE              PIC S9(4)  COMP VALUE -1.
```

```
12.4     01 NUM-IN                    PIC X(13).
12.5     01 NUM-OUT                   PIC S9(9)V99.
12.6     01 NUM-DISP-13               PIC ---------9.99.
12.7
12.8
12.9     01 ACCT-MSTR                    COPY ACCTMSTR.
13
13.1     01 VEND-MSTR                    COPY VENDMSTR.
13.2
13.3     01  DATA-BUF.
13.4         05 FILLER        PIC X(69).
13.5         05 DATA-IN       PIC X(443).
13.6
13.7     01 MENU-DATA       REDEFINES DATA-BUF.
13.8         05 FILLER                  PIC X(69).
13.9         05 SELECT-IN               PIC X.
14           05 SELECT-ERR            PIC X(26).
14.1
14.2     01 VEND-IN       REDEFINES DATA-BUF.
14.3         05 FILLER                  PIC X(69).
14.4         05 VEND-NBR                PIC X(6).
14.5         05 VEND-NBR-ERR            PIC X(26).
14.6         05 VEND-NAME               PIC X(30).
14.7         05 S-ADDRESS               PIC X(30).
14.8         05 S-ADDRESS2              PIC X(30).
14.9         05 S-CITY                  PIC X(20).
15           05 S-STATE                 PIC X(2).
15.1         05 S-ZIP                   PIC X(10).
15.2         05 P-ADDRESS               PIC X(30).
15.3         05 P-ADDRESS2              PIC X(30).
15.4         05 P-CITY                  PIC X(20).
15.5         05 P-STATE                 PIC X(2).
15.6         05 P-ZIP                   PIC X(10).
15.7         05 PHONE-NBR.
15.8          10 PHONE-AC               PIC X(3).
15.9          10 PHONE-EX               PIC X(3).
16            10 PHONE-NO               PIC X(4).
16.1         05 FLAG-1099               PIC X(2).
16.2         05 VEND-CODE               PIC X(2).
16.3         05 VENDOR-STATUS           PIC X(2).
16.4         05 VEND-CODE-ERR           PIC X(26).
16.5         05 VEND-STATUS-ERR         PIC X(26).
16.6
16.7     01 BUDG-IN                   REDEFINES DATA-BUF.
16.8         05 FILLER                  PIC X(69).
16.9         05 ACCT-NBR                PIC X(20).
17           05 BUDG-NBR-ERR            PIC X(26).
17.1         05 BUDG-NY-AMT             PIC X(13).
17.2
17.3     ************************************************
17.4     ***          FORM TABLE LAYOUT:
17.5     ***               SELECTION CHARACTER
17.6     ***               FORM NUMBER
17.7     ***               NEXT FORM NUMBER
17.8     ***               HELP FORM NUMBER
17.9     ***
18       ************************************************
```

```
18.1
18.2    ************************************************
18.3    ***            FORM NAME TABLE LAYOUT:
18.4    ***                FORM NAME (VIEW FORM NAME)
18.5    ***                LENGTH OF DATA FIELDS
18.6    ***
18.7    ************************************************
18.8
18.9
19       01 FORM-TABLE.
19.1         05 FORM-1.
19.2          10 FILLER                  PIC X     VALUE "Z".
19.3          10 FILLER                  PIC 9(4) COMP VALUE  1.
19.4          10 FILLER                  PIC 9(4) COMP VALUE  1.
19.5          10 FILLER                  PIC 9(4) COMP VALUE  14.
19.6         05 FORM-2.
19.7          10 FILLER                  PIC X     VALUE "A".
19.8          10 FILLER                  PIC 9(4) COMP VALUE  2.
19.9          10 FILLER                  PIC 9(4) COMP VALUE  1.
20            10 FILLER                  PIC 9(4) COMP VALUE  13.
20.1         05 FORM-3.
20.2          10 FILLER                  PIC X     VALUE "B".
20.3          10 FILLER                  PIC 9(4) COMP VALUE  3.
20.4          10 FILLER                  PIC 9(4) COMP VALUE  1.
20.5          10 FILLER                  PIC 9(4) COMP VALUE  13.
20.6         05 FORM-4.
20.7          10 FILLER                  PIC X     VALUE "C".
20.8          10 FILLER                  PIC 9(4) COMP VALUE  4.
20.9          10 FILLER                  PIC 9(4) COMP VALUE  1.
21            10 FILLER                  PIC 9(4) COMP VALUE  11.
21.1         05 FORM-5.
21.2          10 FILLER                  PIC X     VALUE "D".
21.3          10 FILLER                  PIC 9(4) COMP VALUE  2.
21.4          10 FILLER                  PIC 9(4) COMP VALUE  1.
21.5          10 FILLER                  PIC 9(4) COMP VALUE  13.
21.6         05 FORM-6.
21.7          10 FILLER                  PIC X     VALUE "E".
21.8          10 FILLER                  PIC 9(4) COMP VALUE  5.
21.9          10 FILLER                  PIC 9(4) COMP VALUE  1.
22            10 FILLER                  PIC 9(4) COMP VALUE  12.
22.1         05 FORM-7.
22.2          10 FILLER                  PIC X     VALUE "7".
22.3          10 FILLER                  PIC 9(4) COMP VALUE  1.
22.4          10 FILLER                  PIC 9(4) COMP VALUE  1.
22.5          10 FILLER                  PIC 9(4) COMP VALUE  12.
22.6         05 FORM-8.
22.7          10 FILLER                  PIC X     VALUE "Z".
22.8          10 FILLER                  PIC 9(4) COMP VALUE  1.
22.9          10 FILLER                  PIC 9(4) COMP VALUE  1.
23            10 FILLER                  PIC 9(4) COMP VALUE  14.
23.1         05 FORM-9.
23.2          10 FILLER                  PIC X     VALUE "Z".
23.3          10 FILLER                  PIC 9(4) COMP VALUE  1.
23.4          10 FILLER                  PIC 9(4) COMP VALUE  1.
23.5          10 FILLER                  PIC 9(4) COMP VALUE  14.
23.6         05 FORM-10.
23.7          10 FILLER                  PIC X     VALUE "Z".
```

```
23.8          10 FILLER                    PIC 9(4) COMP VALUE  1.
23.9          10 FILLER                    PIC 9(4) COMP VALUE  1.
24            10 FILLER                    PIC 9(4) COMP VALUE  14.
24.1
24.2      01 FORM-SPEC-ARRAY             REDEFINES FORM-TABLE.
24.3          05 FORM-SPECS                 OCCURS  10 TIMES.
24.4           10 FORM-ID                   PIC X.
24.5           10 FORM-NBR                  PIC 9(4)  COMP.
24.6           10 FORM-NEXT                 PIC 9(4)  COMP.
24.7           10 FORM-HELP                 PIC 9(4)  COMP.
24.8
24.9      01 FORM-NAME-TABLE.
25            05 FORM-1.
25.1           10 FILLER                    PIC X(15) VALUE "BUDMAINT_MENU  ".
25.2           10 FILLER                    PIC 9(4)  COMP VALUE  96.
25.3          05 FORM-2.
25.4           10 FILLER                    PIC X(15) VALUE "VENDOR_DATA    ".
25.5           10 FILLER                    PIC 9(4)  COMP VALUE 357.
25.6          05 FORM-3.
25.7           10 FILLER                    PIC X(15) VALUE "BANK_MSTR_DATA ".
25.8           10 FILLER                    PIC 9(4)  COMP VALUE 325.
25.9          05 FORM-4.
26             10 FILLER                    PIC X(15) VALUE "BUDGET_LOAD    ".
26.1           10 FILLER                    PIC 9(4)  COMP VALUE 128.
26.2          05 FORM-5.
26.3           10 FILLER                    PIC X(15) VALUE "               ".
26.4           10 FILLER                    PIC 9(4)  COMP VALUE 96.
26.5          05 FORM-6.
26.6           10 FILLER                    PIC X(15) VALUE "               ".
26.7           10 FILLER                    PIC 9(4)  COMP VALUE 96.
26.8          05 FORM-7.
26.9           10 FILLER                    PIC X(15) VALUE "               ".
27             10 FILLER                    PIC 9(4)  COMP VALUE  96.
27.1          05 FORM-8.
27.2           10 FILLER                    PIC X(15) VALUE "               ".
27.3           10 FILLER                    PIC 9(4)  COMP VALUE 96.
27.4          05 FORM-9.
27.5           10 FILLER                    PIC X(15) VALUE "               ".
27.6           10 FILLER                    PIC 9(4)  COMP VALUE 96.
27.7          05 FORM-10.
27.8           10 FILLER                    PIC X(15) VALUE "               ".
27.9           10 FILLER                    PIC 9(4)  COMP VALUE 96.
28            05 FORM-11.
28.1           10 FILLER                    PIC X(15) VALUE "HELP_BANK_DATA ".
28.2           10 FILLER                    PIC 9(4)  COMP VALUE 69.
28.3          05 FORM-12.
28.4           10 FILLER                    PIC X(15) VALUE "HELP_BUDG_LOAD ".
28.5           10 FILLER                    PIC 9(4)  COMP VALUE 69.
28.6          05 FORM-13.
28.7           10 FILLER                    PIC X(15) VALUE "HELP_VENDOR    ".
28.8           10 FILLER                    PIC 9(4)  COMP VALUE 69.
28.9          05 FORM-14.
29             10 FILLER                    PIC X(15) VALUE "BUDMAINT_HELP  ".
29.1           10 FILLER                    PIC 9(4)  COMP VALUE 69.
29.2          05 FORM-15.
29.3           10 FILLER                    PIC X(15) VALUE "HELP_CREDIT_SUP".
29.4           10 FILLER                    PIC 9(4)  COMP VALUE 69.
```

```
29.5
29.6     01 FORM-NAME-ARRAY    REDEFINES FORM-NAME-TABLE.
29.7          05 FORM-NAME-INFO       OCCURS  15 TIMES.
29.8           10 FORM-NAME                PIC X(15).
29.9           10 FORM-DATA-LEN            PIC 9(4)  COMP.
30
30.1
30.2
30.3
30.4
30.5
30.6
30.7     01 FORMAT-CNTL                  COPY FORMTCTL.
30.8
30.9     01 FORMAT-13.
31            05 FILLER                 PIC S9(4)  COMP VALUE 1.
31.1          05 FILLER                 PIC S9(4)  COMP VALUE 13.
31.2          05 FILLER                 PIC S9(4)  COMP VALUE 0.
31.3          05 FILLER                 PIC X      VALUE " ".
31.4          05 FILLER                 PIC X      VALUE SPACES.
31.5          05 D13-FORMAT.
31.6           10 FILLER                PIC S9(4)  COMP VALUE 11.
31.7           10 FILLER                PIC S9(4)  COMP VALUE ZERO.
31.8           10 FILLER                PIC X      VALUE SPACES.
31.9           10 FILLER                PIC X      VALUE "N".
32             10 FILLER                PIC X      VALUE "1".
32.1           10 FILLER                PIC X      VALUE "2".
32.2
32.3
32.4     01 ERROR-ARRAY.
32.5          05 FIELD-ERR          OCCURS 56 TIMES    PIC 9.
32.6     01 FIELD-ZERO       REDEFINES ERROR-ARRAY    PIC X(56).
32.7
32.8     01 FIELD-CNT                 PIC 99  COMP.
32.9     01 FIELD-LOC                 PIC 99  COMP.
33
33.1     ******************************************************************
33.2     $PAGE
33.3
33.4
33.5     PROCEDURE DIVISION .
33.6     000000-MAIN-PART SECTION 01.
33.7     000000-PROGRAM-LOGIC.
33.8          PERFORM 900000-OPEN-PROGRAM.
33.9          PERFORM 100000-READ-LOOP UNTIL LAST-KEY = 8.
34            PERFORM 970000-CLOSE-PROGRAM.
34.1          STOP RUN.
34.2
34.3     100000-READ-LOOP.
34.4          PERFORM 801000-READ-TERM.
34.5          IF LAST-KEY = 0
34.6             NEXT SENTENCE
34.7           ELSE IF LAST-KEY = 1
34.8             PERFORM 881000-KEY-1
34.9           ELSE IF LAST-KEY = 2
35               PERFORM 882000-KEY-2
35.1           ELSE IF LAST-KEY = 3
```

```
35.2            PERFORM 883000-KEY-3
35.3          ELSE IF LAST-KEY = 4
35.4            PERFORM 884000-KEY-4
35.5          ELSE IF LAST-KEY = 5
35.6            PERFORM 885000-KEY-5
35.7          ELSE IF LAST-KEY = 6
35.8            PERFORM 886000-KEY-6
35.9          ELSE IF LAST-KEY = 7
36              PERFORM 887000-KEY-7
36.1          ELSE
36.2            PERFORM 888000-KEY-8.
36.3
36.4        IF CHECK-RESULT NOT = 0  OR LAST-KEY = 5
36.5            NEXT SENTENCE
36.6           ELSE IF LOC-FORM =  1
36.7            PERFORM 101000-EDIT-MENU
36.8          ELSE IF LOC-FORM =  2
36.9            PERFORM 102000-EDIT-A
37            ELSE IF LOC-FORM =  3
37.1            PERFORM 103000-EDIT-B
37.2          ELSE IF LOC-FORM =  4
37.3            PERFORM 104000-EDIT-C
37.4          ELSE IF LOC-FORM =  5
37.5            PERFORM 105000-EDIT-D
37.6          ELSE IF LOC-FORM =  6
37.7            PERFORM 106000-EDIT-E
37.8          ELSE
37.9            MOVE  4 TO CHECK-RESULT.
38
38.1        IF CHECK-RESULT NOT = ZERO
38.2            NEXT SENTENCE
38.3          ELSE IF LOC-FORM =  1
38.4            PERFORM 150000-VALID-MENU
38.5          ELSE IF LOC-FORM =  2
38.6            PERFORM 151000-VALID-A
38.7          ELSE IF LOC-FORM =  3
38.8            PERFORM 152000-VALID-B
38.9          ELSE IF LOC-FORM =  4
39              PERFORM 153000-VALID-C
39.1          ELSE IF LOC-FORM =  6
39.2            PERFORM 155000-VALID-E
39.3          ELSE IF LAST-KEY NOT = 5
39.4            PERFORM 889000-ASK-CONFIRM
39.5          ELSE IF LOC-FORM =  5
39.6            PERFORM 154000-VALID-D
39.7          ELSE
39.8            MOVE 4 TO CHECK-RESULT.
39.9
40
40.1        IF CHECK-RESULT = 3 PERFORM 851000-REFRESH-TERM.
40.2        IF CHECK-RESULT = 4 MOVE 3 TO V-SHOW-CONTROL
40.3         MOVE "Z" TO MODE-FLAG.
40.4        IF CHECK-RESULT = 0 PERFORM 811000-FORM-INITIALIZE.
40.5        IF CHECK-RESULT = 2 OR = 4 OR = 6 PERFORM 852000-NEXT-FORM.
40.6        IF CHECK-RESULT NOT = 9
40.7            PERFORM 804000-SHOW-FORM
40.8            PERFORM 803000-CLEAR-WINDOW.
```

```
40.9         MOVE ZERO TO CHECK-RESULT, V-SHOW-CONTROL.
41
41.1
41.2     101000-EDIT-MENU.
41.3         MOVE ZERO TO CHECK-RESULT.
41.4         PERFORM 805000-VIEW-EDIT.
41.5         PERFORM 807000-GET-BUFFER.
41.6         MOVE SPACES TO SELECT-ERR.
41.7         IF SELECT-IN  = "X" MOVE 8 TO LAST-KEY
41.8            MOVE 9 TO CHECK-RESULT
41.9          ELSE
42              PERFORM 101100-EDIT-MENU-DATA.
42.1
42.2     101100-EDIT-MENU-DATA.
42.3         PERFORM 101110-MENU-SCAN VARYING LOC-FIND FROM 1  BY  1
42.4          UNTIL LOC-FIND >  15
42.5            OR SELECT-IN = FORM-ID (LOC-FIND).
42.6         IF LOC-FIND >  15 PERFORM 101120-MENU-ERROR
42.7          ELSE PERFORM 803000-CLEAR-WINDOW.
42.8
42.9     101110-MENU-SCAN.
43           EXIT.
43.1
43.2     101120-MENU-ERROR.
43.3         MOVE
43.4          "PLEASE SELECT ONE OF THE ABOVE LETTERS AND RE-ENTER"
43.5          TO ERR-MES-BUF.
43.6         MOVE 51  TO LEN-ERR-BUF.
43.7         MOVE  2  TO FIELD-NBR.
43.8         PERFORM 812000-SET-ERROR.
43.9         ADD  1 TO V-NUM-ERRS.
44           MOVE 1 TO CHECK-RESULT.
44.1
44.2
44.3     ************************************************************
44.4     *                    MODE SELECTIONS                       *
44.5     *    A = ADD VENDOR MASTER          N =                    *
44.6     *    B = ADD BANK MASTER            O =                    *
44.7     *    C = ADD NEXT YR.BUDGET         P =                    *
44.8     *    D = EDIT VENDOR MASTER         Q =                    *
44.9     *    E = LOAD BUDGET                R =                    *
45       *    F =                            S =                    *
45.1     *    G =                            T =                    *
45.2     *    H =                            U =                    *
45.3     *    I =                            V =                    *
45.4     *    J =                            W =                    *
45.5     *    K =                            X =  EXIT  PROGRAM     *
45.6     *    L =                            Y =                    *
45.7     *    M =                            Z = MAIN MENU          *
45.8     *                                                          *
45.9     *                                                          *
46       ************************************************************
46.1
46.2
46.3     ************************************************************
46.4     ***              CHECK RESULT  CODES                     ***
46.5     ***     0 = NO ERRORS              5 =                   ***
```

```
46.6    ***     1 = ERRORS, RE-READ          6 = NEXT FORM       ***
46.7    ***     2 = BRING UP NEW FORM        7 =                 ***
46.8    ***     3 = REFRESH TERMINAL         8 =                 ***
46.9    ***     4 = RETURN TO MAIN MENU      9 = EXIT PROGRAM    ***
47      ***                                                      ***
47.1    ************************************************************
47.2
47.3
47.4    ************************************************************************
47.5    ***          FUNCTION  KEY CODES                                      ***
47.6    ***    F1 =  SKIP                    F5  = CONFIRM ENTRY             ***
47.7    ***    F2 = CLEAR (INITIALIZE)       F6  = NEXT FORM                 ***
47.8    ***    F3 = HELP                     F7  = MAIN MENU                 ***
47.9    ***    F4 = REFRESH SCREEN           F8  = EXIT                      ***
48      ***                                                                  ***
48.1    ************************************************************************
48.2
48.3     102000-EDIT-A.
48.4         MOVE ZERO TO CHECK-RESULT.
48.5         MOVE ZERO TO FIELD-ZERO.
48.6         MOVE  16 TO FIELD-CNT.
48.7         PERFORM 805000-VIEW-EDIT.
48.8         PERFORM 807000-GET-BUFFER.
48.9         MOVE SPACES TO VEND-NBR-ERR, VEND-STATUS-ERR.
49           PERFORM 102100-CK-VEND-NBR.
49.1         PERFORM 102200-CK-VEND-CODE.
49.2         PERFORM 102300-CK-VEND-STATUS.
49.3         PERFORM 102400-CK-VEND-1099.
49.4
49.5         PERFORM 809000-PUT-BUFFER.
49.6         PERFORM 805000-VIEW-EDIT.
49.7         IF V-NUM-ERRS NOT = ZERO MOVE 1 TO CHECK-RESULT.
49.8         MOVE ZERO TO FIELD-LOC.
49.9         PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
50
50.1     102100-CK-VEND-NBR.
50.2         MOVE VEND-NBR OF VEND-IN TO ARGUMENT.
50.3         PERFORM 831000-GET-VEND-MSTR.
50.4         IF COND-WORD = 17 NEXT SENTENCE
50.5          ELSE MOVE "INVALID! DUPLICATE NUMBER" TO VEND-NBR-ERR
50.6           MOVE  1 TO FIELD-ERR (2)
50.7           MOVE  1 TO CHECK-RESULT.
50.8
50.9     102200-CK-VEND-CODE.
51           IF VEND-CODE OF VEND-IN = "VN" OR = "VM" OR = "DP"
51.1          NEXT SENTENCE
51.2           ELSE MOVE "INVALID VENDOR CODE!" TO VEND-CODE-ERR
51.3            MOVE  1 TO FIELD-ERR (19)
51.4            MOVE  1 TO CHECK-RESULT.
51.5
51.6     102300-CK-VEND-STATUS.
51.7         IF VENDOR-STATUS OF VEND-IN = "CR" OR = "XX"
51.8          NEXT SENTENCE
51.9           ELSE MOVE "INVALID STATUS CODE!" TO VEND-STATUS-ERR
52              MOVE  1 TO FIELD-ERR (20)
52.1            MOVE  1 TO CHECK-RESULT.
52.2
```

```
52.3     102400-CK-VEND-1099.
52.4          IF FLAG-1099 OF VEND-IN = SPACES OR = "Y "
52.5           NEXT SENTENCE
52.6           ELSE
52.7             MOVE  1 TO FIELD-ERR (18)
52.8             MOVE  1 TO CHECK-RESULT.
52.9
53
53.1
53.2
53.3     103000-EDIT-B.
53.4          MOVE ZERO TO CHECK-RESULT.
53.5          MOVE ZERO TO FIELD-ZERO.
53.6          MOVE  3 TO FIELD-CNT.
53.7          PERFORM 805000-VIEW-EDIT.
53.8          PERFORM 807000-GET-BUFFER.
53.9          PERFORM 103100-CK-VEND-NBR.
54            PERFORM 102200-CK-VEND-CODE.
54.1          PERFORM 102300-CK-VEND-STATUS.
54.2          PERFORM 102400-CK-VEND-1099.
54.3          PERFORM 809000-PUT-BUFFER.
54.4          PERFORM 805000-VIEW-EDIT.
54.5          IF V-NUM-ERRS  NOT = ZERO
54.6             MOVE  1 TO CHECK-RESULT.
54.7          MOVE ZERO TO FIELD-LOC.
54.8          PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
54.9
55       103100-CK-VEND-NBR.
55.1          IF NEW-FLAG = ZERO NEXT SENTENCE
55.2           ELSE IF VEND-NBR OF VEND-IN = VEND-NBR OF VEND-MSTR
55.3             MOVE  1 TO NEW-FLAG
55.4           ELSE MOVE ZERO TO NEW-FLAG.
55.5          MOVE VEND-NBR OF VEND-IN TO ARGUMENT.
55.6          PERFORM 831000-GET-VEND-MSTR.
55.7          IF COND-WORD = ZERO AND NEW-FLAG = 1
55.8             NEXT SENTENCE
55.9           ELSE IF COND-WORD = ZERO
56               PERFORM 103110-SETUP-VENDOR
56.1           ELSE
56.2             MOVE "NON-EXISTENT VENDOR NUMBER!" TO VEND-NBR-ERR
56.3             MOVE  1 TO FIELD-ERR (2)
56.4             MOVE  1 TO CHECK-RESULT.
56.5
56.6     103110-SETUP-VENDOR.
56.7          MOVE CORR VEND-MSTR TO VEND-IN.
56.8          MOVE  1 TO NEW-FLAG, CHECK-RESULT.
56.9
57       104000-EDIT-C.
57.1          MOVE ZERO TO CHECK-RESULT.
57.2          MOVE ZERO TO FIELD-ZERO.
57.3          MOVE  3 TO FIELD-CNT.
57.4          PERFORM 805000-VIEW-EDIT.
57.5          PERFORM 807000-GET-BUFFER.
57.6     *    MOVE SPACES TO ERROR DISPLAY FIELDS
57.7     *    PERFORM EDIT ROUTINES
57.8          PERFORM 809000-PUT-BUFFER.
57.9          PERFORM 805000-VIEW-EDIT.
```

```
58              IF V-NUM-ERRS  NOT = ZERO
58.1               MOVE  1 TO CHECK-RESULT.
58.2            MOVE ZERO TO FIELD-LOC.
58.3            PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
58.4
58.5        105000-EDIT-D.
58.6            MOVE ZERO TO CHECK-RESULT.
58.7            MOVE ZERO TO FIELD-ZERO.
58.8            MOVE  3 TO FIELD-CNT.
58.9            PERFORM 805000-VIEW-EDIT.
59              PERFORM 807000-GET-BUFFER.
59.1       *    MOVE SPACES TO ERROR DISPLAY FIELDS
59.2       *    PERFORM EDIT ROUTINES
59.3            PERFORM 809000-PUT-BUFFER.
59.4            PERFORM 805000-VIEW-EDIT.
59.5            IF V-NUM-ERRS  NOT = ZERO
59.6               MOVE  1 TO CHECK-RESULT.
59.7            MOVE ZERO TO FIELD-LOC.
59.8            PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
59.9
60          106000-EDIT-E.
60.1            MOVE ZERO TO CHECK-RESULT.
60.2            MOVE ZERO TO FIELD-ZERO.
60.3            MOVE  3 TO FIELD-CNT.
60.4            PERFORM 805000-VIEW-EDIT.
60.5            PERFORM 807000-GET-BUFFER.
60.6       *    MOVE SPACES TO ERROR DISPLAY FIELDS
60.7       *    PERFORM EDIT ROUTINES
60.8            PERFORM 809000-PUT-BUFFER.
60.9            PERFORM 805000-VIEW-EDIT.
61              IF V-NUM-ERRS  NOT = ZERO
61.1               MOVE  1 TO CHECK-RESULT.
61.2            MOVE ZERO TO FIELD-LOC.
61.3            PERFORM 813000-SET-ERROR-FIELDS FIELD-CNT TIMES.
61.4
61.5
61.6        150000-VALID-RECORD SECTION 03.
61.7        150000-VALID-MENU.
61.8            MOVE 2 TO CHECK-RESULT.
61.9            MOVE LOC-FIND TO LOC-FORM.
62              MOVE ZERO TO NEW-FLAG.
62.1
62.2        151000-VALID-A.
62.3
62.4        152000-VALID-B.
62.5
62.6        153000-VALID-C.
62.7
62.8        154000-VALID-D.
62.9
63          155000-VALID-E.
63.1
63.2
63.3
63.4
63.5        800000-UTILITIES SECTION 02.
63.6
```

```
63.7     801000-READ-TERM.
63.8         CALL "VREADFIELDS" USING VIEW-COM.
63.9         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
64
64.1     802000-PUT-WINDOW.
64.2         CALL "VPUTWINDOW" USING VIEW-COM ERR-MES-BUF LEN-ERR-BUF.
64.3
64.4     803000-CLEAR-WINDOW.
64.5         MOVE SPACES TO ERR-MES-BUF.
64.6         PERFORM 802000-PUT-WINDOW.
64.7
64.8
64.9     804000-SHOW-FORM.
65           CALL "VSHOWFORM" USING VIEW-COM.
65.1         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
65.2
65.3
65.4     805000-VIEW-EDIT.
65.5         CALL "VFIELDEDITS" USING VIEW-COM.
65.6
65.7     806000-FINISH-FORM.
65.8         CALL "VFINISHFORM" USING VIEW-COM.
65.9
66       807000-GET-BUFFER.
66.1         CALL "VGETBUFFER" USING VIEW-COM DATA-BUF V-DBUF-LEN.
66.2         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
66.3
66.4     809000-PUT-BUFFER.
66.5         CALL "VPUTBUFFER" USING VIEW-COM DATA-BUF V-DBUF-LEN.
66.6         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
66.7
66.8     810000-GET-FORM-FILE.
66.9         MOVE 0  TO V-REPEAT-OPT.
67           MOVE 0  TO V-NF-OPT.
67.1         CALL "VGETNEXTFORM" USING VIEW-COM.
67.2         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
67.3
67.4     811000-FORM-INITIALIZE.
67.5         MOVE 65  TO V-WINDOW-ENH.
67.6         CALL "VINITFORM" USING VIEW-COM.
67.7     *   ADD INDIVIDUAL FORMS INITIALIZATION HERE AS REQ.
67.8         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
67.9         PERFORM 803000-CLEAR-WINDOW.
68           MOVE  1 TO LAST-RESULT.
68.1
68.2     812000-SET-ERROR.
68.3         CALL "VSETERROR" USING VIEW-COM FIELD-NBR ERR-MES-BUF
68.4          LEN-ERR-BUF.
68.5         IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
68.6
68.7     813000-SET-ERROR-FIELDS.
68.8         ADD 1  TO FIELD-LOC.
68.9         IF FIELD-ERR (FIELD-LOC) = 1
69              CALL "VSETERROR" USING VIEW-COM FIELD-LOC ERR-MES-BUF
69.1            NO-MESSAGE
69.2              IF V-STATUS NOT = ZERO
69.3                PERFORM 992000-VIEW-ERROR
```

```
69.4              ELSE
69.5                NEXT SENTENCE
69.6           ELSE
69.7             NEXT SENTENCE.
69.8
69.9       814000-CONFIRM-READ.
70             CALL "IMMVREADFIELDS" USING VIEW-COM.
70.1           IF V-STATUS NOT = ZERO  PERFORM 992000-VIEW-ERROR.
70.2
70.3
70.4       820000-SPACE-NUMBER.
70.5           MOVE FORMAT-13 TO FORMAT-CNTL.
70.6           CALL "CAPE'ENTRY" USING FORMAT-CNTL NUM-IN NUM-OUT.
70.7           IF CFIELD-ERR (1) = ZERO AND CENTRY-ERR = ZERO
70.8            MOVE NUM-OUT TO NUM-DISP-13
70.9             ELSE MOVE 1 TO CHECK-RESULT.
71
71.1       830000-GET-ACCT-MSTR.
71.2           MOVE "ACCOUNT-MSTR;" TO DSET-NAME.
71.3           CALL "DBGET" USING FBASE DSET-NAME MODE7 STATUS-AREA
71.4            ALL-ITEMS ACCT-MSTR ARGUMENT.
71.5           IF COND-WORD NOT = ZERO AND NOT = 17
71.6              PERFORM 991000-STATUS-CK
71.7            ELSE
71.8              NEXT SENTENCE.
71.9
72         831000-GET-VEND-MSTR.
72.1           MOVE "VENDOR-MSTR;" TO DSET-NAME.
72.2           CALL "DBGET" USING FBASE DSET-NAME MODE7 STATUS-AREA
72.3            ALL-ITEMS VEND-MSTR ARGUMENT.
72.4           IF COND-WORD = ZERO OR = 17
72.5              NEXT SENTENCE
72.6            ELSE
72.7              PERFORM 991000-STATUS-CK.
72.8
72.9       841000-DB-LOCK.
73             CALL "DBLOCK" USING FBASE DSET-NAME MODE3 STATUS-AREA.
73.1           IF COND-WORD NOT = ZERO PERFORM 991000-STATUS-CK.
73.2
73.3       842000-DB-UNLOCK.
73.4           CALL "DBUNLOCK" USING FBASE DSET-NAME MODE1 STATUS-AREA.
73.5           IF COND-WORD NOT = ZERO PERFORM 991000-STATUS-CK.
73.6
73.7
73.8
73.9       851000-REFRESH-TERM.
74             PERFORM 980000-CLOSE-TERM.
74.1           PERFORM 902000-OPEN-TERM.
74.2           MOVE 3 TO V-SHOW-CONTROL.
74.3           PERFORM 811000-FORM-INITIALIZE.
74.4
74.5       852000-NEXT-FORM.
74.6           IF CHECK-RESULT =  4 MOVE  1  TO LOC-FORM.
74.7           MOVE FORM-NBR (LOC-FORM) TO LOC-FORM-NAME.
74.8           MOVE FORM-NAME (LOC-FORM-NAME) TO V-NFNAME.
74.9           PERFORM 810000-GET-FORM-FILE.
75             PERFORM 811000-FORM-INITIALIZE.
```

```
75.1
75.2     853000-EDIT-ERROR.
75.3          CALL "VERRMSG" USING VIEW-COM ERR-MES-BUF LEN-ERR-BUF
75.4           LEN-ERR-MES.
75.5          IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
75.6          CALL "VPUTWINDOW" USING VIEW-COM ERR-MES-BUF LEN-ERR-BUF.
75.7
75.8     854000-PUT-TITLE.
75.9     *    MOVE 1 TO FIELD-NBR.
76       *    CALL "VPUTFIELD" USING VIEW-COM FIELD-NBR TITLE-BUF
76.1     *     TITLE-LEN ACT-FIELD-LEN NEXT-FIELD-NBR.
76.2     *    IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
76.3
76.4
76.5
76.6     860000-START-HELP.
76.7          MOVE FORM-HELP (LOC-FORM) TO LOC-FORM-NAME.
76.8          MOVE FORM-NAME (LOC-FORM-NAME)  TO V-NFNAME.
76.9
77       861000-HELP-DISPLAY.
77.1          PERFORM 810000-GET-FORM-FILE.
77.2          PERFORM 854000-PUT-TITLE.
77.3          PERFORM 804000-SHOW-FORM.
77.4          CALL "VREADFIELDS" USING VIEW-COM.
77.5          IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
77.6          IF LAST-KEY = 8
77.7             MOVE  9 TO CHECK-RESULT
77.8           ELSE IF LAST-KEY = 7
77.9             MOVE  4 TO CHECK-RESULT
78            ELSE IF LAST-KEY = 4
78.1             MOVE  3 TO CHECK-RESULT
78.2           ELSE
78.3             MOVE 2 TO CHECK-RESULT.
78.4
78.5
78.6     865000-INITIAL-VENDOR.
78.7     *    MOVE VEND-NBR OF VEND-MSTR TO VEND-NBR OF VEND-IN.
78.8     *    MOVE VEND-NAME OF VEND-MSTR TO VEND-NAME OF VEND-IN.
78.9     *    PERFORM 809000-PUT-BUFFER.
79       *
79.1
79.2     870000-UPDATE-ACCT-MSTR.
79.3          CALL "DBUPDATE" USING FBASE DSET-NAME MODE1
79.4           STATUS-AREA ALL-ITEMS ACCT-MSTR.
79.5          IF COND-WORD = ZERO NEXT SENTENCE
79.6           ELSE PERFORM 991000-STATUS-CK.
79.7
79.8
79.9
80       881000-KEY-1.
80.1          MOVE 1 TO CHECK-RESULT.
80.2          MOVE "INVALID KEY SELECTED, IGNORED" TO ERR-MES-BUF.
80.3          MOVE 29 TO LEN-ERR-BUF.
80.4          PERFORM 802000-PUT-WINDOW.
80.5
80.6     882000-KEY-2.
80.7          MOVE 1 TO CHECK-RESULT.
```

```
80.8         PERFORM 811000-FORM-INITIALIZE.
80.9
81       883000-KEY-3.
81.1         PERFORM 860000-START-HELP THRU 861000-HELP-DISPLAY.
81.2
81.3     884000-KEY-4.
81.4         MOVE 3 TO CHECK-RESULT.
81.5
81.6     885000-KEY-5.
81.7         IF LAST-RESULT = ZERO
81.8            PERFORM 814000-CONFIRM-READ
81.9            MOVE  5 TO LAST-KEY
82              MOVE ZERO TO CHECK-RESULT
82.1         ELSE
82.2            PERFORM 890000-INVALID-CONFIRM.
82.3
82.4     886000-KEY-6.
82.5         MOVE 6 TO CHECK-RESULT.
82.6         MOVE FORM-NEXT (LOC-FORM ) TO LOC-FORM.
82.7
82.8
82.9     887000-KEY-7.
83           MOVE "Z" TO MODE-FLAG.
83.1         MOVE 4 TO CHECK-RESULT.
83.2
83.3     888000-KEY-8.
83.4         MOVE 9 TO CHECK-RESULT.
83.5
83.6     889000-ASK-CONFIRM.
83.7         MOVE
83.8          "VALID RECORD, PUSH CONFIRM KEY (F5) TO POST AS SHOWN"
83.9           TO ERR-MES-BUF.
84           MOVE 52  TO LEN-ERR-BUF.
84.1         PERFORM 802000-PUT-WINDOW.
84.2         MOVE  1 TO CHECK-RESULT.
84.3         MOVE ZERO TO LAST-RESULT.
84.4
84.5     890000-INVALID-CONFIRM.
84.6         MOVE  1 TO CHECK-RESULT.
84.7         MOVE
84.8          "INVALID USE OF CONFIRM KEY!  CONFIRM NOT REQUESTED!"
84.9          TO ERR-MES-BUF.
85           MOVE  52 TO LEN-ERR-BUF.
85.1         PERFORM 802000-PUT-WINDOW.
85.2
85.3     ****************************************************************
85.4
85.5     900000-START-STOP SECTION 51.
85.6     900000-OPEN-PROGRAM.
85.7         DISPLAY "VIEW/COBOL LAYOUT PROGRAM  VERS. 0.01".
85.8         PERFORM 901000-OPEN-DATA-BASE.
85.9         PERFORM 902000-OPEN-TERM.
86           PERFORM 903000-OPEN-VFORM.
86.1         PERFORM 904000-START-MENU.
86.2
86.3     901000-OPEN-DATA-BASE.
86.4         CALL "DBOPEN" USING FBASE PASSWORD MODE1 STATUS-AREA.
```

```
86.5          IF COND-WORD = ZERO
86.6             NEXT SENTENCE
86.7           ELSE
86.8             PERFORM 991000-STATUS-CK
86.9             STOP RUN.
87
87.1      902000-OPEN-TERM.
87.2          MOVE ZERO TO V-STATUS, V-LANGUAGE,
87.3           VIEW-MODE, LAST-KEY, V-NUM-ERRS, V-REPEAT-OPT
87.4           V-NF-OPT.
87.5          CALL "VOPENTERM" USING VIEW-COM TERM-FILE.
87.6          IF V-STATUS = ZERO
87.7             NEXT SENTENCE
87.8           ELSE
87.9             PERFORM 992000-VIEW-ERROR
88               CALL "VCLOSEFORMF" USING VIEW-COM
88.1             DISPLAY ERR-MES-BUF  " STOPPING RUN! "
88.2             PERFORM 990000-STOP-PAR.
88.3
88.4      903000-OPEN-VFORM.
88.5          MOVE "BUDGFORM,BUDGET,PROGLIB" TO V-FILE-NAME.
88.6          CALL "VOPENFORMF" USING VIEW-COM V-FILE-NAME.
88.7          IF V-STATUS = ZERO
88.8             NEXT SENTENCE
88.9           ELSE
89               PERFORM 992000-VIEW-ERROR
89.1             CALL "VCLOSETERM" USING VIEW-COM
89.2             DISPLAY ERR-MES-BUF  " STOPPING RUN! "
89.3             PERFORM 990000-STOP-PAR.
89.4
89.5      904000-START-MENU.
89.6          MOVE  1  TO LOC-FORM.
89.7          PERFORM 852000-NEXT-FORM.
89.8          PERFORM 804000-SHOW-FORM.
89.9          MOVE ZERO TO CHECK-RESULT.
90
90.1
90.2      970000-CLOSE-PROGRAM.
90.3          PERFORM 980000-CLOSE-TERM.
90.4          PERFORM 981000-CLOSE-VFORM.
90.5          PERFORM 990000-STOP-PAR.
90.6
90.7      980000-CLOSE-TERM.
90.8          CALL "VCLOSETERM" USING VIEW-COM.
90.9          IF V-STATUS NOT = 0 PERFORM 992000-VIEW-ERROR.
91
91.1      981000-CLOSE-VFORM.
91.2          CALL "VCLOSEFORMF" USING VIEW-COM.
91.3          IF V-STATUS NOT = ZERO PERFORM 992000-VIEW-ERROR.
91.4
91.5
91.6      990000-STOP-PAR.
91.7          CALL "DBCLOSE" USING FBASE DSET-NAME MODE1 STATUS-AREA.
91.8          STOP RUN.
91.9
92
92.1      991000-STATUS-CK.
```

```
92.2          PERFORM 980000-CLOSE-TERM.
92.3          PERFORM 981000-CLOSE-VFORM.
92.4          CALL "DBEXPLAIN" USING STATUS-AREA.
92.5          PERFORM 990000-STOP-PAR.
92.6
92.7      992000-VIEW-ERROR.
92.8
92.9          CALL "VERRMSG" USING VIEW-COM ERR-MES-BUF LEN-ERR-BUF
93             LEN-ERR-MES.
93.1          DISPLAY BELL "VIEW ERROR!!!".
93.2          DISPLAY ERR-MES-BUF.
93.3          DISPLAY BELL "PROGRAM TERMINATED DUE TO ABOVE ERROR!!!".
93.4          PERFORM 990000-STOP-PAR.
93.5
```

PASCAL? ADA?? PEARL!!!

Process and Experiment Automation Realtime Language in Industrial and University's Environment

PEARL on HP3000/HP1000 Networks

Klaus Rebensburg
Technical University of Berlin (West Germany)

INTRODUCTION

Steadily increasing software costs and the inherent risks computer automation projects made it necessary to replace the obsolete technology of assembler programming by structured programming in a high-order real-time language. At present, PEARL is the world's most powerful and advanced high-order real-time language used for industrial process automation.

PEARL has been developed for application engineers. In comparison to system implementation languages like Concurrent PASCAL, MODULA or ADA, PEARL is the only real-time language which the user can learn quickly and apply readily. In contrast to (Process-) FORTRAN, PEARL is a homogeneous language which e.g. has built-in language elements for process input/output and for task-scheduling. Thus PEARL offers a higher degree of portability than any other language for process automation.

At the Technical University of Berlin PEARL is implemented on a HP3000 Series II and HP1000 F in autonomous versions and in the university's process-control network as cross-version between HP3000→HP1000 computers.

PASCAL?

PASCAL offers good tools for structured programming. For a lot of applications the user needs tasking facilities and process-input/output. Built-in language elements are the best solution to program such problems. PASCAL does not have these scheduling and I/O elements.

ADA??

ADA is a high order programming language for so-called "embedded systems." ADA was developed in order and under control of the Department of Defense (DOD) and is supposed to become a widely spread programming language. ADA offers good tools for structured programming. Especially the attempt to define the compiler environment is a good approach for cost-effective software-production.

Multiprogramming facilities are provided with a minimal set of basic operations. No attempt is made to define special features covering the large range of input/output applications. The language facilities are designed in a way that the user has to provide I/O packages to define special process-I/O features.

That means there is no standardization for process-I/O in ADA. Until now there is no official ADA-compiler system running. Software producers have to wait — how long? There are some problems to make ADA learnable for application engineers, especially the so-called rendesvouz-concept might be difficult to map it on real technical processes. There are some other problems to develop ADA programs on minicomputers as ADA is designed for cross-software development on big computer systems.

How many years must a HP-user wait for ADA3000 or ADA1000?? Remember — PASCAL was defined in 1971, now in the 1980s we will look forward to use the official PASCAL on HP3000. ADA was defined 1979.

PEARL !!!

- Algorithmic language elements
- Abstract data types
- Modular program structure
- Real-time language elements
- Description of the hardware-configuration / Input/output language elements
- Who uses PEARL
- Standardization of PEARL
- Availability of PEARL for Hewlett-Packard Computers

What Are the Characteristics of PEARL?

PEARL offers all language characteristics which the user needs in order to solve his industrial automation problems. Some software houses use PEARL as implementation language for all kinds of greater software-problems, like database-management, big software-packets.

Algorithmic Language Elements

The language elements for the formulation of algorithms and procedures correspond to the state of the art of modern programming languages (e.g., PASCAL).

Algorithmic elements can be written as declarations of procedures, functions, and user-defined operators (!) and tasks (!). (The last two features are e.g. not provided by PASCAL.)

Tasks are elements for parallel execution.

List of PEARL statements:

```
variable := expression;              ( assignation )
GOTO   identifier;
CALL   proc_identifier;              ( procedure call )
RETURN (expression);                 ( return from procedure,function )
INDUCE signal_identifier;            ( raise an exception )

ON signal_identifier: statement;     ( reaction on exception )

BEGIN  declarations                  ( block )
       statements
END;

IF condition                         ( if-then-else construct )
   THEN statements
   ELSE statements
FIN;

CASE expression                      ( case construct )
  ALT statements
  ...
  OUT statements
FIN;

FOR variable                         ( for-while-loop construct )
  FROM expr BY expr TO expr
  WHILE condition
    REPEAT declaration_list
           statements
    END;
```

Abstract Data Types

The modern concept of data types in PEARL enables the user to define problem oriented, composite data types and new operators. These abstract data types permit a great number of checks at compile time and contribute to a refined modular program structure (strong typing).

```
basic types: FIXED                   ( fixed )
             FLOAT                   ( floating point )
             BIT                     ( bit )
             CHAR                    ( character )

             REF                     ( pointer )
             CLOCK, DURATION         ( real-time !! )
             SEMA,  BOLT             ( semaphores, bolts !! )

structures:  array  (dynamic)
             STRUCT
             bit chain
             DATION                  ( standard and user-defined
                                       peripherals )

             (structures can be nested)


allocation of variables:
             at address determined by compiler
             RESIDENT attribute indicates fast access
```

```
access to variables:
            by name or reference

standard operators:
            many standard operators are defined in the langage,
            e.g., ABS, LWB, TOFIXED, TOFLOAT, TOBIT, TOCHAR,
            and CSHIFT.
```

Modular Program Structure

A PEARL program is composed of separately compilable modules with exactly defined interfaces. This structure greatly facilitates communication between the members of a project team and supports the modular composition of complex program-systems.

Example:

```
MODULE (reportwriter); /* written by programmer A */
  SYSTEM;
    /* contains description of configuration */
  PROBLEM;
    /* specification of imported objects ;
       declaration of tasks,procedures,operators,data,types,dations */
    ...;
MODEND; /* end of module reportwriter */

MODULE ( brewery_control); /* written by programmer B */
  PROBLEM;
      ...
MODEND; /* end of module brewery_control */
```

Real-time Language Elements / Multiprogramming Facilities of PEARL

For programming task scheduling, PEARL contains real-time language elements, which can be learnt and applied easily.

Example:

```
AFTER 5 SEC ALL 7 SEC DURING 106 MIN ACTIVATE relay PRIORITY 5;
(5 seconds after the execution of this statement the computing
 process 'relay' is activated with priority 5 every 7 seconds
 for a total period of 106 minutes)


extended_time_specification ACTIVATE task; ( scheduling )
TERMINATE task;
SUSPEND   task;
time_spec CONTINUE task;
time_spec RESUME   task;
PREVENT task;

operations on semaphores:          ( synchronization, communication )
  REQUEST, RELEASE
operations on bolt variables:      ( multiple reader-writer problems )
  RESERVE, FREE, ENTER, LEAVE
operations on interrupts:          ( interrupt-handling )
  DISABLE, ENABLE, TRIGGER

  WHEN interrupt_identifier task_control_statement;

operations on signals:             ( exceptions )
  ON signal: statement;            ( react upon exception )
  INDUCE signal;                   ( raise an exception )

Example: At 16:00:30 RESUME task; ( delay )

         WHEN interrupt_id AFTER 10 SEC EVERY 20 MIN UNTIL 15:20:00
         ACTIVATE task_id;
```

Description of the Hardware-configuration / Input/output Language Elements

In the System-Division of PEARL the hardware configuration, especially process peripherals, can be described independently of the special hardware realisation by user defined identifiers. This capability greatly enhances documentation value and portability of PEARL programs.

Example:

```
MODULE (demo); /* this module contains a PEARL - program */

SYSTEM;

  display: stdio(1); /* display is the identifier used in the
                        program for standard-I/O. stdio(1) is a
                        system defined name for the HP3000
                        implementation                          */

  disk   : disc(3);  /* the user disk */

  engine : prog_cont;/* engine is the identifier used in the
                        program for 16 Bit I/O. The HP3000 knows
                        it as programmable controller interface  */

PROBLEM;

  /* we start with the specification of the peripherals */

  SPECIFY display   DATION INOUT ALPHIC     CONTROL(ALL),
          disk      DATION IN    ALL DIM(,) CONTROL(ALL),
          engine    DATION OUT   BASIC;

  /* now we use these peripherals */
  STARTtest: TASK;

  DECLARE on        INV BIT(16) INIT ('0000000000000111'),
          off       INV BIT(16) INIT ('0000000000000001');

  /* let's switch the engine */

  SEND on TO engine;
  AFTER 2 MIN RESUME;     /* continue 2 minutes later */
  SEND off TO engine;
  PUT `we have stopped the engine' TO display by SKIP,A(30),SKIP;

  END; /* end of task STARTtest */
  ...
MODEND; /* end of module */
```

Another example for PEARL I/O:

```
TAKE pressure FROM pressure_sensor;

SEND open TO valve;      /* output of the value `open' to device
                            `valve' */
```

Input and output features: Data-stations (DATIONs), generalizing real or virtual peripherals or I/O channels. Interfaces, mapping data-stations with different properties onto each other to offer the possibility to define formatting routines (objects of type CONTROL).

Most of these PEARL systems don't use Hewlett-Packard computers (?). PEARL compilers are offered by many (mostly European) computer manufacturers and software houses.

An object of type DATION represents in general a set of one to four channels:

Data	channel (transfer values of PEARL objects)
Control	channel (transfers values of type CONTROL)
Interrupt	channel (signals events of type INTERRUPT)
Signal	channel (signals events of type SIGNAL)

Who Uses PEARL?

PEARL is already widely used (with applications mostly in W-Germ Process computer projects in many different areas have been sucessfully programmed with PEARL.

The following table is a survey of PEARL activities (1980):

Field of Application	Number of PEARL Systems
Metal Processing, Rolling Mills	41
Power Distribution	36
Power Generation	5
Raw Materials, Chemical Industry	21
Water Supply	14
Other Industrial Applications	21
Mail Order Houses, Warehouses	12
Television, Transport, Aerospece Applications	21
Development and Education	32
	204

Following computer systems are available with PEARL compilers: AEG 80-20, AEG 80-60, Data General NOVA, DP1000/1500 by BBC PDP 11 family, Krupp-Atlas EPR 1100/1300/1500, Hewlett-Packard HP1000 and HP3000, Interdata 7/32, INTEL 8086, LSI 11, Micronova MUC161, MUDAS 432, MULBY 3, NORD 10 by Norsk Data, Digital Equipment PDP11 family, PDP 11/34, Siemens 300,310,404/3, VAX 11/780, Z80, MOTOROLA MC68000.

Standardization of PEARL

The standardization of PEARL ensures its uniformity. The draft standard DIN 66253 part 1 "BASIC PEARL" has been available since 1978. Draft standard DIN 66253 part 2 "FULL PEARL" followed in August 1980. Parallel to these activities, PEARL has been submitted to ISO for international standardization (TC97/SC5/WG1). For these purposes PEARL was described completely with petri-nets and attributed grammars (formal syntax and semantics).

Availability of PEARL for HP Computers

The Technical University of Berlin uses PEARL in three different ways. The university's real-time process-control computer network consists of 1 HP3000 series II 512 KByte, 16 HP1000 computers (M,E,F) and 30 microcomputer systems. The central computer HP3000 is used for program-development, documentation, cross-software, statistics, graphics, education, whereas the decentral HP1000 systems are used for process-control applications.

a) PEARL3000 is the autonomous PEARL Programming System on HP3000. It is used for program development and training courses. Typical courses are visited by 18 persons each, 3-4 teachers 7 terminals, 5 days mixed theory, practical work, discussions. Participants are EDP leaders, engineers, programmers from industry and university.

The Technical University Berlin is responsible for design and execution of the official PEARL courses, offered by the VDI (Verein der Ingenieure) Germany and by the PEARL Association Duesseldorf.

In connection to the compiler system a PEARL Testsystem on language level is available. It simulates, e.g., tasking, checks for deadlocks, I/O. PEARL3000 is running under MPE4 on HP3000 Series II/III computers.

b) PEARL1000 is the autonomous PEARL Programming System on HP1000. It is used for program development, training courses, real-time and process-control applications. Most of the PEARL1000 implementation was developped by the Technical University of Berlin. The implementation was sponsored by the Ministry of Research and Technology of (West-) Germany. PEARL1000 is running under RTE4B with HP1000 F Computers (256 KBytes)

c) PEARL3000/1000 is the Cross PEARL System for HP3000/HP1000 networks. Compilation, interface-check, relocation is done on HP3000 with the obvious advantage that during the coding phase many programmers can work simultaneously.

The cross-version supports small HP1000 configurations, core-resident systems which are autonomous in process-control and loosely coupled with V.25 lines to a HP3000 multiplexor for transfer of relocated code to HP1000 and process-data to HP3000. PEARL3000/1000 supports HP1000 computers under RTEM3 operating systems.

The HP3000 has proven to be a good development computer system for all kinds of other computers. We not only use it for HP1000 computers but also for more than 14 different types of microprocessors.

In all three cases the same PEARL compiler is used. Code generation in each case is adapted to the target-computer. Of course runtime routines and operating system kernel are tailored to the different HP standard operating systems.

No change of MPE or RTE operation system was necessary for the implementation. (Cross-version has some minor changes of RTEM3 operating system.)

Use and handling of the PEARL Compilation Systems is easy and comparable to FORTRAN3000 or other HP subsystems.

REFERENCES

1. DIN 66253 Part 1 "BASIC PEARL" (draft standard, language: English). 1978 Beuth Verlag GmbH Berlin 30, W-Germany.
2. DIN 66253 Part 2 "FULL PEARL" (draft standard, language: English). 1980 Beuth Verlag GmbH Berlin 30, W-Germany.
3. Werum, Wulf: Windauer, Hans; PEARL Process and Experiment Automation Realtime Language. 1978 Vieweg Verlag Braunschweig; Book, which describes HP3000/HP1000 PEARL language. Language: English and German
4. Kappatsch, A.; Mittendorf, H.; Rieder, P. PEARL Systematic

description for the application engineer. R. Oldenbourg Verlag Muenchen Wien 1979. Language: German

5. Kappatsch, A., PEARL Survey of language features. Kernforschungszentrum Karlsruhe KfK-PDV 141 August 1977. Language: English
6. Martin, T.; PEARL AT THE AGE OF THREE. 4th International Conference on Software-Engeneering, September 17-19, 1979 Munich, Germany IEEE No.79 CH1479-5G. Language: English.
7. Martin, T.; PEARL AT THE AGE OF FIVE. Updated Version, published in *Computers in Industry*, Vol. 3, Number 2, 1981. Language: English.
8. Hommel, G.; Experience with PEARL in Industrial Applications. VDE-Congress, Berlin 6.-9.10. 1980. Language: German.
9. Windauer, H.; Development and Implementation of Portable Compilers for Realtime Languages. Proceedings of Real-Time Data '79 Berlin Oct. 1979. Language: English.
10. Martin,T.; Realtime Programming Language PEARL Concepts and Characteristics. Proceeding 2nd Computer Software and Applications Conf., Chicago, 1978, pp 301-306 IEEE Cat.No.78CH 1338-3C.
11. Brinkkoetter, H., Groessler,J., Nagel, K., Nebel, H., Kneuer,E., Rebensburg, K.; PEARL on Hewlett-Packard Computers Implementation and Demonstration. 27.4.1981 Berlin. Language: German.
12. Rebensburg, K.; Real-time Computing with the Process-Control Computer Network of the Technical University Berlin. Language: English.
13. Brinkkoetter, H., Nagel, K., Nebel, H., Rebensburg, K.; Systematic Programming with PEARL. PEARL Training Course Handbook, VDI/PEARL Association, 1981 Berlin. Language: German.
14. PEARL Association; PEARL-RUNDSCHAU. Official bimonthly publication by the PEARL Association Verein. Contains contributions of PEARL applications, software-houses, educational aspects, scientific applications, PEARL News etc. Language: German.

All information about PEARL can be obtained from the author and from:

PEARL Association
Graf-Recke-Strasse 84
4000 Duesseldorf 1

P.S. PEARL information is also available in Spanish, Portuguese, Chinese, French and Serbocroatic languages.

Application Design Implications of PASCAL/3000 Dynamic Variable Allocation Support or How to Use the HEAP

Steven Saunders
Information Networks Division
Hewlett-Packard Company
Cupertino, California

ABSTRACT

This paper is intended to introduce PASCAL/3000's dynamic variable allocation support. This introduction is used as a basis for a discussion of application design issues relating to the PASCAL/3000 support environment. The details of the PASCAL/3000 implementation which are needed to interface to existing applications are presented.

The dynamic allocation of variables is provided through memory-management routines operating on an area called the HEAP. The PASCAL language and support environment provide a means of explicitly controlling the allocation and deallocation of variables in the HEAP. These features provide the programmer with the ablity to implement designs which combine a high degree of adaptability and reliability.

Approaches for making design and implementation decisions based on the capabilities of dynamic variable allocation in PASCAL/3000 are presented. Examples of good and bad use of these capabilities are discussed. The knowledge gained in the design and implementation of the PASCAL/3000 compiler is used as the basis for this discussion.

A good working knowledge of dynamic variable allocation can provide the application designer with insights into producing designs that are more adaptable to the user's needs. The application or systems programmer can use knowledge of the PASCAL HEAP to more effectively implement quality software.

INTRODUCTION

The programming language PASCAL has a large number of features that are provided by other programming languages available on the HP3000. There are also some significant features that are unique to PASCAL, such as strong type checking and dynamic variable support features.

The strong type checking feature, which will not be covered in this paper, permits improved compiler verification of data abstractions and module interfaces.

This paper discusses the dynamic variable support features which facilitate the creation of programs which are adaptable and versatile, yet simple.

The discussion of PASCAL's dynamic variable support features is broken into three sections:
The first section introduces the concepts of dynamic variable support, and explains the terminology used in this paper. The concepts are presented in the context of the PASCAL/3000 implementation. The defining occurrence of each new word of terminology is capitalized in the text.

The second section reexamines these concepts in the context of application design. This context is used to contrast design approaches employing PASCAL's dynamic variable support to more "conventional" approaches used with languages like FORTRAN or COBOL.

The third section discusses the interaction of the language features, implementation details, and design approaches. That interaction is examined in the context of problems that application designers and implementors might encounter with the use of dynamic variables.

I. DEFINITION AND USE OF DYNAMIC VARIABLES IN PASCAL/3000

The DYNAMIC VARIABLE support feature of PASCAL allow a program to allocate on demand any number of global variables, irrespective of the program's block structure. Dynamic variables are global, in contrast to static variables that are local to the block in which they are declared. This reflects the most important aspect of dynamic variables, the separation of the declaration and allocation of storage for a variable. The PASCAL/3000 implementation provides this basic dynamic variable support along with several extensions.

Sample Program

The following sample program is used throughout this paper to illustrate PASCAL/3000 syntax. The program builds a linked list with dynamic variables, and then traverses and prints the list. The numbers contained in the comments to the left of each line are used to reference that line in the text of the paper. The lines containing "{HP}" near the right margin are extensions defined by the Hewlett-Packard PASCAL standard.

```
{LINE}
{ 0} $HEAP_DISPOSE ON, HEAP_COMPACT OFF$
{ 1} PROGRAM IUG_Example (OUTPUT);

{ 2} TYPE
{ 3}    Pointer_Type = ^ Record_Type;

{ 4}    Record_Type = RECORD
{ 5}       Integer_Field : INTEGER;
{ 6}       Pointer_Field : Pointer_Type;
{ 7}    END;

{ 8} CONST                                                    {HP}
{ 9}    Integer_Const = 27;
{10}    Pointer_Const = NIL;                                  {HP}

{11}    Record_Const = Record_Type [                          {HP}
{12}       Integer_Field : 0,                                 {HP}
{13}       Pointer_Field : Pointer_Const                      {HP}
{14}    ];                                                    {HP}

{15} VAR
{16}    Integer_Var : INTEGER;
{17}    Pointer_Var : Pointer_Type;
{18}    Record_Var  : Record_Type;

{19}    BEGIN
{20}    Integer_Var := Integer_Const;
{21}    Pointer_Var := Pointer_Const;                         {HP}
{22}    Record_Var  := Record_Const;                          {HP}

{23}    NEW (Pointer_Var);
{24}    Record_Var .Pointer_Field := Pointer_Var;

{25}    WHILE Integer_Var > 0 DO
{26}       BEGIN
{27}       Pointer_Var^ .Integer_Field := Integer_Var;
{28}       Integer_Var := PRED(Integer_Var);

{29}       NEW (Pointer_Var^ .Pointer_Field);
{30}       Pointer_Var^ Pointer_Field^ := Record_Const;       {HP}
{31}       Pointer_Var := Pointer_Var^ .Pointer_Field;
{32}       END;

{33}    Pointer_Var := Record_Var .Pointer_Field;
{34}    WHILE Pointer_Var <> NIL DO
{35}       BEGIN
{36}       WRITELN (Pointer_Var^ .Integer_Field);
{37}       Pointer_Var := Pointer_Var^ .Pointer_Field;
{38}       END;
{39}    END.
```

Definition and Use of "Conventional" Static Variables

STATIC VARIABLES can be characterized as named storage areas that exist only during the execution of the procedure or function that in which they are declared. Their existence can be determined by simply looking at a program listing. This static nature means that the number of static variables and the size of each static variable are fixed when the program is compiled. Thus, the storage for static variables can be allocated when their declarations are processed. This is how PASCAL implementations handle static variables; storage for them is allocated when the block containing their declarations is entered.

The fixed number and size of static variables forces designers and implementors to wastefully reserve storage for rarely used and/or large data structures (e.g., data structures for year-end versus month-end processing). However, static variables are very useful for holding frequently computed results.

The variables declared in lines 16 through 18 in the sample program are global static variables. GLOBAL STATIC VARIABLES are the same as any other (local) static variables, except that they are allocated before a program begins execution and exist as long as it is executing.

Aspects of Dynamic Variables

The separation of declaration and allocation of storage for dynamic variables has one key implication: the number of dynamic variables is NOT fixed when the program is compiled. The size of an individual dynamic variable is fixed when the program IS compiled, just as it is for a static variable, but the number of dynamic variables can change. The changable number of dynamic variables enables application designers and implementors to provide storage for rare and/or large data structures without much effort.

A dynamic variable is defined as being pointed to by a pointer which can only point to a single unique type, which is the type of the dynamic variable. Line 3 of the sample program shows the declaration of the type "Pointer_Type" that points to dynamic variables of the type "Record_Type". Line 17 shows the declaration of "Pointer_Var," a static variable of this type. "Pointer_Var" 's value can be used to access a dynamic variable, but it is not itself a dynamic variable.

Line 6 shows a component, "Pointer_Field", of the structured type "Record_Type", whose value can be used to access a dynamic variable of the same structured type. This form of declaration can be employed to build linked data structures.

Usage Of Dynamic Variables

Dynamic variables must be explicitly allocated and deallocated by a program. Thus, the existence of dynamic variables depends on the dynamic (execution) behavior of a program. The number and arrangement of dynamic variables cannot be determined statically (by simply looking at a program listing). In contrast to static variables, dynamic variables do not have actual names. Dynamic variables' "names" are just unique values generated by the dynamic storage allocation mechanism.

Dynamic variables are allocated in PASCAL by calling the system-supplied procedure NEW. This procedure selects a storage area for the requested type of dynamic variable from an area called the HEAP. Lines 23 and 29 of the sample program show the use of New to allocate dynamic variables of the type "Record_Type". New sets the values of "Pointer_Var" and "Pointer_Field", in lines 23 and 29, respectively.

Dynamic variables are deallocated in PASCAL/3000 by one of two mechanisms. The first, standard to all PASCAL implementations, is the system-supplied procedure DISPOSE. This procedure deallocates a single dynamic variable at a time.

The second mechanism is a Hewlett-Packard PASCAL extension. This extension provides two additional system-supplied procedures, Mark and Release. The procedure MARK creates a generic pointer value that describes the state of the HEAP. The STATE OF THE HEAP can be characterized as a temporal reference point. All allocations of dynamic variables can be unambiguously classified as occurring either before or after this reference point. A GENERIC POINTER VALUE can be the value of any pointer, irrespective of the type of dynamic variable it is declared to point to. Any pointer having a generic pointer value does not point to any dynamic variable. The procedure RELEASE uses a generic pointer value created by a previous call to Mark to restore the state of the HEAP. This results in the deallocation of all dynamic variables allocated after the reference point denoted by the generic pointer value. Put very simply Release deallocates all dynamic variables allocated after the corresponding call to Mark.

PASCAL supplies a generic pointer value NIL which can and should be used to indicate pointers that are not currently pointing to any allocated dynamic variable.

The value of any dynamic variable can be inspected or modified by DEREFERENCING any pointer pointing to that dynamic variable. The up-arrow "^" or the at-sign "@" are used to syntactically denote dereferencing. Line 30 of the sample program shows an assignment to the dynamic variable pointed to by the "Pointer_Field" component of the dynamic variable pointed to by the static variable "Pointer_Var". All dereferences take this form of starting the dereferencing sequence with some static pointer variable (e.g., "Pointer_Var" in line 27). Lines 31, 33, 36, and 37 show the use of dereferencing to inspect the value of a component of a dynamic variable.

Implementation of Dynamic Variables in PASCAL/3000

The HEAP area of any PASCAL/3000 program is the DL-DB area of the stack segment of the process executing that program. Static variables are stored in the DB-S area of the same stack segment. The value of a pointer to an allocated dynamic variable is the word address of the first word of that dynamic variable.

The generic pointer values created by Mark are of a form known only to the PASCAL/3000 implementation. The generic pointer value Nil is equal to the word address +32767, the theoretical upper limit of a HP3000 stack segment.

The allocation of dynamic variables in PASCAL/3000 can involve one or two methods of "finding" storage space for a dynamic variable. The basic method essentially amounts to moving DL futher away from DB to get the needed storage area.

The second method requires that the compiler option "HEAP_DISPOSE ON" be specified (e.g., line 0 of sample program). If it is, a free list of deallocated areas is searched for the first area large enough to store a dynamic variable of the type requested. If this search fails then the basic method is used.

The deallocation of dynamic variables in PASCAL/3000 by the Dispose procedure depends on compiler options as shown in the decision table below:

$HEAP_DISPOSE ON$	True	True	False
$HEAP_COMPACT ON$	True	False	-
Do nothing			X
Insert area into free list	X	X	
Combine w/ adjacent free areas	X		

It should be noted that the "Do nothing" action in the table above is what the Dispose procedure does in many PASCAL implementations. The "Insert area into free list" and "Combine w/adjacent free areas" actions require the system to have one (1) word of overhead for each dynamic variable allocated.

The deallocation of dynamic variables in PASCAL/3000 by the Release procedure amounts to moving DL to where it was when Mark was called. The operation of this procedure is independent of all compiler options.

Limitations of Dynamic Variables in PASCAL/3000

The basic limitations of dynamic variables in PASCAL/3000 are that the number of variables that can be allocated is limited, and that each has its size fixed when the program is compiled. The limited number is the result of the HEAP residing entirely within the stack segment, which is limited to 32,767 words. That is felt to be the most realistic design choice simply because of the large overhead associated with randomly accessing data not stored in the stack segment. All implementations have some sort of upper limit on their dynamic storage, and this is the upper limit that makes sense on the HP3000. This limitation makes it necessary to design and implement most applications with some use of dynamic variable deallocation.

The second limitation is common to all PASCAL implementations and can be overcome by proper design.

II. DYNAMIC VARIABLE DESIGN CONSIDERATIONS AND IMPLICATIONS

The most effective way to use dynamic variables in an application is to consider their use when designing the application. The two key aspects of this approach are decomposing a program's data into indivisible data items and choosing between alternative allocation/deallocation models. The flexibility and expressiveness of dynamic variables are also important design considerations.

Constrasting Concepts of What a Data Item Is

The data items used in applications written in languages without dynamic variables tend to be thought of as counters, temporaries, buffers, and tables. The first three of these can be easily implemented as static variables. But data items used as tables can have limited flexibility if implemented as static arrays, excessive implementation complexity if they are implemented as adjustable or virtual arrays, and poor performance if implemented as files.

The problem is not the limitations of these methods of implementing tables. The problem is simply that thinking of data items as tables does not always reflect the reality of application's intended function. Instead of thinking of a data item as a monolithic table, a designer could decompose it into small pieces, each piece representing a "chunk" of information. Each of these pieces would be related to the other pieces in well-defined ways. This way of organizing information lends itself to the dynamic variable approach. The benifits of this approach are that the number of dynamic variables is not fixed as is the number of elements in a static array, and that well-defined relationships can be easily implemented by pointers.

This is not to imply that all tables should be replaced

with structures composed of linked dynamic variables. But any "table" data items that must support dynamic insertion and/or deletion of "element" data items are good candidates for implementation with dynamic variables.

The application implementor could, independently of the designer, convert any static table to a structure composed of dynamic variables. This will work acceptably in some cases, but fail in others (e.g., converting a randomly-accessed table to a simple linked list). The designer can, as part of the decomposition process, make suggestions on the use of dynamic variables. But an even more promising benefit of using dynamic variables is that the whole structure of an application could be improved. A designer that understands how dynamic variables permit an adaptive implementation will no doubt create more versatile designs.

Models of Dynamic Variable Allocation/Deallocation

The limitations of the PASCAL/3000 implementation require that most applications employing dynamic variable allocation also employ some form of dynamic deallocation. This requirement can be met by considering allocation/deallocation models as part of the application design. The choice of the proper model can maximize the number of available dynamic variables while minimizing the system overhead. Four basic models will be presented here, in order of increasing flexibility and overhead.

1. *Fire Sale Model*

The FIRE SALE MODEL, the simplest model, does not require any system support of deallocation. The name of this model is used in analogy to the nonreturnable nature of items purchased in a fire sale. An application employing the fire sale model can maintain its own free lists, one list for each type of dynamic variable. This model is useful for designing applications that will allocate new data items, but seldom, if ever, need to deallocate them. An example of an application employing the model is a PERT analysis program. The program builds the PERT graph and then analyzes it. The fire sale model makes use of the following compiler options and system-supplied procedures:

```
Compiler Options    - $HEAP_DISPOSE OFF$
System Procedures   - New
```

2. *Stack Model*

The STACK MODEL of deallocation requires that the system can save and restore the state of the HEAP. An application employing the stack model can maintain its own free lists. This model is useful for designing applications that will allocate new data items in groups and then will deallocate the groups in reverse order of allocation. An example of an application employing this model is the PASCAL/3000 compiler. The compiler groups the allocation of data items based on the block structure of the source program, processing and then deallocating the innermost block first. The stack model makes use of the following compiler options and system-supplied procedures:

```
Compiler Options    - $HEAP_DISPOSE OFF$
System Procedures   - New
                      Mark/Release
```

3. *Pool Model*

The POOL MODEL of deallocation requires that the system is able to place deallocated dynamic variables in a pool of free storage and allocate new dynamic variables from this pool. An application employing the pool model should not maintain its own free lists. This model is useful for designing applications that will allocate and deallocate data items more or less simultaneously. An example of an application employing this model would be a shop floor simulation program. The simulation would allocate, process, and deallocate job, task, and event data items in an interleaved manner. The PASCAL/3000 implementation will support two variations of this model. The first does not combine contiguous free storage blocks, and will only work well when very few unique types of dynamic variables are used. This variation of the pool model makes use of the following compiler options and system-supplied procedures:

```
Compiler Options    - $HEAP_DISPOSE ON$
                      $HEAP_COMPACT OFF$

System Procedures   - New
                      Dispose
```

The second variation does combine contiguous free storage blocks, and will work well in all cases, but suffers more execution time overhead. This variation of the pool model makes use of the following compiler options and system-supplied procedures:

```
Compiler Options     - $HEAP_DISPOSE ON$
                       $HEAP_COMPACT ON$
System Procedures    - New
                       Dispose
```

4. *Hybrid Model*

The HYBRID MODEL of deallocation requires that the system support both the stack and pool models of deallocation. An application employing the hybrid model should not maintain its own free lists. This model is useful for designing aplications that will allocate and deallocate data items simultaneously as well as in groups. An example of an application employing this model would be a natural language query processor. The program would allocate and deallocate data items to build a model of the world and the necessary queries. When a set of queries is completed, the world model data items would be deallocated as a group. The hybrid model makes use of the following compiler options and system-supplied procedures:

```
Compiler Options     - $HEAP_DISPOSE ON$
                       $HEAP_COMPACT ON$
System Procedures    - New
                       Mark/Release
                       Dispose
```

The User's Input CAN Determine The Number of Data Items

The major advantage that the use of dynamic variables can offer to the designer is adaptability. For example, someone designing an information retrieval system for employee data would not have to make arbitrary decisions about the maximum number of employee dependents that the system could handle. Rather, the designer would simply describe the employee and dependent data items and their relationship. With dynamic allocation, the required number would be allocated when the application was run.

The flexibility of dynamic variables was a key tool in designing the PASCAL/3000 compiler. The compiler was designed not to have any arbitrary limits save for the HEAP size limit. Thus, a programmer need not, for example, be worried that Case statements could have no more than 1023 case label values. This general freedom from limits without increased complexity could greatly enhance the usability and useful life of many applications.

Natural Implementation of Algorithms and Data Structures

The implementation of many algorithms and the data structures they operate on is significantly easier with dynamic variables. This is simply because many algorithms for peforming operations on complex data structures were designed with dynamic variables in mind (e.g., insertion to, deletion from, and searching of height-balanced binary trees). Thus designers, and especially implementors, can take advantage of the work of others to achieve better results without having to "re-invent the wheel." As mentioned before, the results of decomposing an application's data items can be readily expressed with dynamic variables.

Direct Representation of Data Item Relationships

The representation of the relationship between two or more static variables can only be described by the logic of the program. The relationship between two or more dynamic variables can be partially represented by the declaration of pointers as components of the dynamic variables. Thus, much of the relationship information for dynamic variables can be maintained inside the variable, while the same information for static variables is maintained outside of the variables. All this, plus the added adaptability resulting from the data item decomposition design approach make dynamic variables a superior way to represent data item relationships.

III. DYNAMIC VARIABLE PITFALLS TO BE AVOIDED

Dynamic variables can provide improvements in software quality, but they can also be used in ways that can seriously degrade quality. These pitfalls can take the following forms:

- A design using a bad choice of allocation/deallocation model.
- Interfacing with external routines that do not respect the integrity of the HEAP area.
- Design or implementation flaws resulting in attempts to access deallocated variables.

- Deliberate or accidential modification of a pointer by accessing the wrong variant of a record.
- Simply attempting to allocate more variables than can fit into a HP3000 stack segment.

While these are not all the possible pitfalls, experience shows they are the most common.

Poor Selection of Allocation/Deallocation Model

The selection of an allocation/deallocation model that is poorly matched to the application design can increase the complexity or reduce the dynamic variable capacity of the application program. If a program employs a fire sale or stack model, uses a great many dynamic variable types and maintains free lists for each of them, then the complexity of the program must increase and more storage may be wasted in free lists than would be used by the system overhead from employing a pool or hybrid model. The solution would be to switch from the fire sale or stack model to either the pool or hybrid model (e.g., use $HEAP_DISPOSE ON$).

The reverse could also be true, that only a few dynamic variable types would need free lists, each containing a few deallocated variables. The solution would then be to switch to a model that used the "HEAP_DISPOSE OFF" compiler option.

Using External Routines That Alter DL-DB Area

The PASCAL/3000 implementation assumes it has total control over the DL-DB area. But many library or utility routines implemented in SPL/3000 make free use of this area (e.g., VPlus/3000 and DSG/3000 previously used this area). The solution was for PASCAL/3000 to provide the intrinsics GETHEAP and RTNHEAP that function just like New and Dispose. These intrinsics can only solve the problem if the routines were designed, or can be modified, to request storage in independent chunks whose individual location is unimportant. There is one further constraint on using this solution with either the stack or hybrid allocation/deallocation models: all areas that an external routine allocates after Mark is called must be deallocated before the corresponding call to Release (e.g., a correct sequence is Mark, GetHeap, RtnHeap, Release).

Accessing Deallocated Dynamic Variables

Dynamic variables that have been deallocated either by Dispose or Mark/Release cannot be accessed. That is the rule, but given that pointer values in PASCAL/3000 are implemented as word addresses, any pointer to a deallocated dynamic variable that has not been modified still points to "something"! Some form of this DANGLING POINTER PROBLEM exists in almost all PASCAL implementations (e.g., it does not exist for implementations that only support the fire sale model). The software failures (bugs) caused by this problem range from bounds violations to obscure, seemingly random, failures in totally unrelated parts of a program. Clearly, at least from the experience of implementing the PASCAL/3000 compiler, this problem must be designed out, not debugged out. The solution to this problem is very application-dependent. It represents the dynamic variable pitfall that the designer should be most concerned about. Below is an incomplete list of several partial solutions that can be incorporated into applications as the designer sees fit.

1. Never have more than one pointer to any dynamic variable.
2. Never have pointer components of dynamic variables point to other dynamic variables allocated after successive calls to Mark.
3. Doubly-link all data structures so that all pointers to a dynamic variable can be set to Nil before the variable is deallocated.
4. Maintain a count of the number of pointers to a dynamic variable in that variable and never deallocate it if the count is greater than 1.
5. Study, document, and analyze the deallocation portions of an application very carefully.
6. Never deallocate any dynamic variable.

Pointers in the Variant Parts of Records

The dangling pointer problem's first cousin is the CLOBBERED POINTER PROBLEM. The problem can occur only when a pointer is a component of the variant part of a record and some component of another variant is modified. This is the worst of the variant record problems because the destruction can spread with very little trace of the source of the bug. For example, an ordinal component of one variant is modified and then a pointer in another variant is dereferenced to modify the dynamic variable it points to. This dynamic variable may now be the record length component of a file control block. The next Read call for this file will fail with an strange error.

The solution is to always use tag fields when declaring variant record types and always check the tag field before inspecting or modifing any component of the associated variant. A simpler but less-widely-useful solution is to never declare any pointer components in any variant part.

Allocating Too Many Dynamic Variables

If dynamic variables are used to implement very versatile applications, then these applications can be made to allocate more dynamic variables than will fit into a stack segment. This HEAP OVERFLOW condition can only be relieved by the deallocation of some dynamic variables. The condition can be prevented by reducing the number or the size of dynamic variables that need to be allocated. These two solutions, reacting to and preventing HEAP overflow, are explored below.

The first requirement of any design that attempts to

handle HEAP overflow is that the application must be notified of the condition. This is provided by PASCAL/3000 through the MPE library trap mechanism. The application designates a library trap handler procedure which is called when an allocation attempt causes a HEAP overflow. The trap handler procedure can take whatever action is deemed appropriate by the designer. For example, when the PASCAL/3000 compiler detects a HEAP overflow, it simply reports the condition and terminates processing.

The design flexibility of preventing a HEAP overflow is much greater than reacting to it, but it can increase application complexity. The approaches for preventing HEAP overflow fall into two broad classes: improving allocation/deallocation efficiency, and allocating some dynamic variables outside the HEAP.

The improvement of allocation/deallocation efficiency depends on reducing the lifespans, peak number, and sizes of dynamic variables used by an application. Reducing the lifespans can usually be best accomplished by using individual deallocation (Dispose) instead of group deallocation (Mark/Release). Reducing the peak or maximum number of dynamic variables allocated is very application-dependent and requires modelling alternative designs. Reducing the size of dynamic variables can be accomplished by impoving the decomposition of the data items (safe way) and/or by allocating the smallest variant needed in each case (dangerous way).

The approach of allocating dynamic variables outside the HEAP results in increased application complexity because much of the work performed by system-supplied procedures and language syntax must be duplicated in the application. The most flexible approach is to adopt coding conventions that permit migration of dynamic variables from inside to outside the HEAP with little modification to the application. The sample program shown below illustrates a coding convention that allows dynamic variables to be allocated inside or outside of the HEAP. The main advantage of this coding convention is that it localizes any changes to three procedures and one function for each dynamic variable type (e.g., _New, _Dispose, _Modify, & _Access). The main disadvantage of this convention is that it necessitates a function call, a procedure call, and moving the value of an entire dynamic variable five (5) times just to modify a single component of it. This convention basically replaces dereferences with procedure and function calls. The sample shows identical operations on two dynamic variable types, prefixed "type1_" and "type2_", one allocated in a file and the other allocated in the HEAP.

```
PROGRAM Allocate_Outside_Heap;
TYPE
   type1_Ptr = 0 .. 32000;
   type1_Rec = RECORD
      data : BOOLEAN;
   END;

   type2_Ptr = ^ type2_Rec;
   type2_Rec = RECORD
      data : BOOLEAN;
   END;
CONST
   type1_Nil   = 0;
   type1_Const = type1_Rec [data: FALSE];

   type2_Nil   = NIL;
   type2_Const = type2_Rec [data: FALSE];
VAR
   type1_Heap : RECORD
      Heap_Limit : type1_Ptr;
      Heap_Store : FILE OF type1_Rec;
   END;

   type1_Ptr1, type1_Ptr2: type1_Ptr;
   type2_Ptr1, type2_Ptr2: type2_Ptr;


   PROCEDURE type1_New (VAR Ptr: type1_Ptr);
      BEGIN
      WITH type1_Heap DO
```

```
      BEGIN
      Heap_Limit := SUCC (Heap_Limit);
      Ptr := Heap_Limit;
      END;
   END;

PROCEDURE type2_New (VAR Ptr: type2_Ptr);
   BEGIN
   NEW (Ptr);
   END;


PROCEDURE type1_Dispose (VAR Ptr: type1_Ptr);
   BEGIN
   Ptr := type1_Nil;
   END;

PROCEDURE type2_Dispose (VAR Ptr: type2_Ptr);
   BEGIN
   DISPOSE (Ptr);
   Ptr := type2_Nil;
   END;


FUNCTION type1_Access (Ptr: type1_Ptr): type1_Rec;
VAR
   type1_Temp1 : type1_REC;

   BEGIN
   WITH type1_Heap DO
      BEGIN
      ASSERT (Ptr <> type1_Nil, 0);
      ASSERT (Ptr <= Heap_Limit, 1);
      READDIR (Heap_Store, Ptr, type1_Temp1);
      type1_Access := type1_Temp1;
      END;
   END;

FUNCTION type2_Access (Ptr: type2_Ptr): type2_Rec;
   BEGIN
   type2_Access := Ptr^;
   END;


PROCEDURE type1_Modify (Ptr: type1_Ptr; Rec: type1_Rec);
   BEGIN
   WITH type1_Heap DO
      BEGIN
      ASSERT (Ptr <> type1_Nil, 0);
      ASSERT (Ptr <= Heap_Limit, 1);
      WRITEDIR (Heap_Store, Ptr, Rec);
      END;
   END;

PROCEDURE type2_Modify (Ptr: type2_Ptr; Rec: type2_Rec);
   BEGIN
   Ptr^ := Rec;
   END;
```

```
PROCEDURE Example1_Procedure (Ptr1, Ptr2: type1_Ptr);
VAR
   type1_Temp1 : type1_Rec;

   BEGIN
   type1_Temp1 := type1_Access (Ptr1);
   WITH type1_Access (Ptr2) DO
      BEGIN
      (* omitted *)
      END;
   type1_Modify (Ptr1, type1_Temp1);
   type1_Modify (Ptr2, type1_Access (Ptr1));
   END;

PROCEDURE Example2_Procedure (Ptr1, Ptr2: type2_Ptr);
VAR
   type2_Temp1 : type2_Rec;

   BEGIN
   type2_Temp1 := type2_Access (Ptr1);
   WITH type2_Access (Ptr2) DO
      BEGIN
      (* omitted *)
      END;
   type2_Modify (Ptr1, type2_Temp1);
   type2_Modify (Ptr2, type2_Access (Ptr1));
   END;


BEGIN
WITH type1_Heap DO
   BEGIN
   Heap_Limit := type1_Nil;
   OPEN (Heap_Store);
   END;

type1_New (type1_Ptr1);
type1_Modify (type1_Ptr1, type1_Const);
type1_New (type1_Ptr2);
type1_Modify (type1_Ptr2, type1_Const);
Example1_Procedure (type1_Ptr1, type1_Ptr2);
type1_Dispose (type1_Ptr1);
type1_Dispose (type1_Ptr2);

type2_New (type2_Ptr1);
type2_Modify (type2_Ptr1, type2_Const);
type2_New (type2_Ptr2);
type2_Modify (type2_Ptr2, type2_Const);
Example2_Procedure (type2_Ptr1, type2_Ptr2);
type2_Dispose (type2_Ptr1);
type2_Dispose (type2_Ptr2);
END.
```

SUMMARY

This paper introduced dynamic variables as supported by the PASCAL/3000 implementation. The concepts and models needed to use dynamic variables in designing more adaptive applications were covered. Finally, some possible pitfalls designers and implementors using dynamic variables might encounter were discussed. The interested reader might consider the following two publications.

for experts:

Pascal/3000 Program Language Reference Manual
Hewlett-Packard Company
Part No. 32106-900001

for beginners:

Programming in PASCAL with PASCAL/1000
Peter Grogono
Addison-Wesley Publishing Company Inc.

Special thanks to Wendy Peikes for her editoral suggestions.

Process Sensing and Control

Nancy Kolitz
Hewlett-Packard Company
Cupertino, California

I. INTRODUCTION

Various MPE intrinsics on the HP3000 allow a user to create processes, to obtain information about them, and to control them. This paper will describe the process sensing and control capabilities available to a user, through illustrations and examples. The paper will also introduce a new intrinsic, PROCINFO, currently being developed by the MPE lab.

II. WHAT IS A PROCESS?

All user programs run as processes under MPE. A process is the unique execution of a program by a particular user at a particular time, and is the entity within MPE which can accomplish work. A process is also the mechanism which allows system resources to be shared and a user's code to be executed. Each process consists of a private data stack and code segments, shared by all processes executing the same program.

As the system is brought up, the Progenitor (PROGEN) is the first process created by MPE. One of the various system processes that Progen creates is the User Controller Process (UCOP), which creates a User Main Process (UMAIN) as a session or job logs on. Then when a user (or job) runs a program, a User Son of Main (USONM) process is created. If other processes are subsequently created from this program, User processes are established. (See Figure 1.)

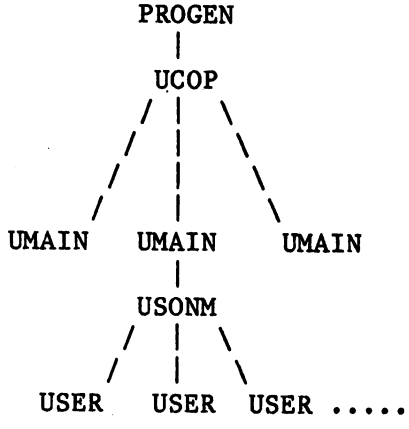


Figure 1

A process will be in one of two states once it has been created: Wait or Active. If it is in a wait state, it is waiting for some event (I/O, RIN acquistion, etc.) to occur before it will run again. If it is in an active state, the process is running or ready to run.

A standard MPE user has no control over his processes. The operating system creates, controls, and kills the processes for the user. However, if the user's program has Process Handling (PH) capability, it can, to some degree, manage its own processes. In fact, it can even control processes in its family tree.

III. PROCESS CREATION

In MPE, there are two intrinsics that a user with PH capability can use to create a process: CREATE and CREATEPROCESS.

The intrinsic CREATE will load a program into virtual memory, create a new process, initialize the stack's data segment, schedule the process to run, and return the process identification (PIN) number to the process requesting the creation. Once the process is established, it will have to be activated by the creating process. The command syntax is:

```
                BA      BA       I   IV    LV      IV      IV
CREATE(progname,entryname,pin,param,flags,stacksize,dlsize,
          IV       LV         IV      O-V
     maxdata,priorityclass,rank).
```

The last parameter RANK (in the CREATE intrinsic) is not used by the intrinsic and is only there for compatibility with previous versions of MPE.

CREATEPROCESS is the other intrinsic that can be used for creating processes. Its format is:

```
                  I    I      BA        IA       LA       O-V
CREATEPROCESS(error,pin,progname,itemnums,items).
```

The parameter ITEMNUMS indicates the options to be applied in creating the new process, and the parameter ITEMS provides the necessary information to be used for each option specified in ITEMNUMS.

With CREATEPROCESS, a son may be activated immediately upon creation or may be activated as a process is with CREATE (via the ACTIVATE intrinsic). A user may also specify an entry point into a program, define $STDIN and $STDLIST to be any file other than the defaults (the defaults are the creating father's $STDIN and $STDLIST), control stack size, and control the process' priority queue. Some of these can also be done with CREATE.

The example that follows illustrates the intrinsic CREATEPROCESS. It will create a process, indicate that the father should be awakened upon completion of the son, and then activate the new process.

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

BEGIN
          <<CREATEPROCESS example>>

INTEGER ERROR, PIN;
BYTE ARRAY EXAMPLE(0:7) := "EXAMPLE ";
INTEGER ARRAY OPTNUMS(0:10);
LOGICAL ARRAY OPTIONS(0:10);

intrinsic CREATEPROCESS,TERMINATE;

          <<set up options>>
OPTNUMS(0) := 3;           OPTIONS(0) := 1;
OPTNUMS(1) := 10;          OPTIONS(1) := 3;
OPTNUMS(2) := 0;     <<terminator>>

CREATEPROCESS(ERROR,PIN,EXAMPLE,OPTNUMS,OPTIONS);

if <> then TERMINATE;

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

When calling MPE intrinsics, a good programming practice is to check the condition code returned, and the error parameter, if one is used. In the case of CREATEPROCESS, if the condition code is less than zero the process was created, but some event occurred to cause the operating system to give a warning to the creator. If the condition code is greater than zero, an error has occurred and the process was not created. If the error occurred because of a file system problem (error number returned is 6), a user can use the intrinsic FCHECK with a parameter of zero to obtain more information as to why the process creation failed.

IV. SENSING PROCESSES

Each process in MPE has a large amount of information about it that can be useful, providing a process can access it. There are various intrinsics that will return this information once a process has been created. However, a program must have PH capability to use these intrinsics.

A user may determine the PIN number of the process that created it via the intrinsic FATHER. Its syntax is:

```
      I
pin := FATHER.
```

Once again, a programmer should check the condition code that was returned. In this case, it will tell what type of process the father is. Through specific codes, it will specify whether the father is a system process, a user main process, or a user process.

To obtain the PIN number of any of his son processes, a program may use the intrinsic GETPROCID. The command is:

```
 I                    IV
pin := GETPROCID(numson).
```

The parameter NUMSON is a integer value that specifies which son a father wants to know the PIN number. For example, if a father has created three sons and wants to know the PIN number of the second son, he will supply GETPROCID with a parameter of two.

The WHO intrinsic provides the access mode and attributes of the user running a program. The file access capabilities (save file (SF), ability to access nonsharable devices (ND), etc.), user attributes (OP, SM, etc) and user capabilities (PH, DS, etc) can be obtained. Also information about the user, his logon group name and account name, his home group, and the logical device of his input file may be returned. The command syntax for WHO is:

```
      L      D        D       BA     BA     BA     BA      L       O-V
WHO(mode,capability,lattr,usern,groupn,acctn,homen,termn).
```

The intrinsic GETORIGIN will return, to a reactivated process, the origin of its activation. The value returned will specify if the PIN was activated from a suspended state by a father, a son, or another source (interrupt or timer). GETORIGIN looks like:

```
 I
source := GETORIGIN.
```

Other information about a son or father may be obtained from the intrinsic GETPROCINFO. Its format is:

```
 D                      IV
statinfo := GETPROCINFO(pin).
```

A double word is passed back giving the process' priority number and priority queue, its activity state (active or waiting), its suspension condition and source of next activation, and the origin of its last activation. The process number, passed as a parameter, specifies which process you want information about. If PIN=0, then information is returned for the father; otherwise, the information is for a son process.

A new intrinsic currently under development in the MPE lab is called PROCINFO. This intrinsic returns general information about processes that is currently unavailable, unless you have privileged mode capability. It should simplify some of the uses of process related intrinsics because a large amount of information may be retrieved in one call to PROCINFO. Its command syntax is:

```
                 I     I     IV      I      BA          O-V
PROCINFO(error1,error2,pin[,itemnum1,item1]
                          [,itemnum2,item2]
                          [,itemnum3,item3]
                          [,itemnum4,item4]
                          [,itemnum5,item5]
                          [,itemnum6,item6]).
```

This intrinsic is formatted similar to FFILEINFO in order to maintain ease of use and extensibility. It can return to a program the process number of the process itself, its father, all its sons, and all its descendants. It can also supply information about the number of descendants and generations in a family tree, the name of a program that a specified process is running, the process' state, and the process' priority number.

The first error word is used to return the type of error incurred when executing the intrinsic. The second error word returns the index of the offending item number. The program name is returned in a byte array that is a minimum of twenty eight bytes long. It is in the format of <filename.group.account>.

The following example will help to illustrate the use of the PROCINFO intrinsic:

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

BEGIN  <<procinfo example>>

  INTEGER ERROR1, ERROR2, PIN;

  BYTE ARRAY ITEMVAL1 (0:1),
             ITEMVAL2 (0:1),
             ITEMVAL3 (0:1),
             ITEMVAL4 (0:1),
             ITEMVAL5 (0:1);

  INTEGER ITEMNUM1,ITEMNUM2,ITEMNUM3,ITEMNUM4,ITEMNUM5;


  INTRINSIC PROCINFO;

  PIN := 0;              <<seek information about ourselves>>
  ITEMNUM1 := 1;     <<request our pin #>>
  ITEMNUM2 := 3;     <<how many sons we have>>
  ITEMNUM3 := 4;     <<how many descendants we have>>
  ITEMNUM4 := 2;     <<pin number of our father>>
  ITEMNUM5 := 5;     <<how many generations we have>>

  PROCINFO (ERROR1, ERROR2, PIN, ITEMNUM1, ITEMVAL1,
                                 ITEMNUM2, ITEMVAL2,
                                 ITEMNUM3, ITEMVAL3,
                                 ITEMNUM4, ITEMVAL4,
                                 ITEMNUM5, ITEMVAL5);

  IF <> THEN GO PROCERROR;


  .  .  .  .

  PROCERROR:
    <<print message and error number>>
    RETURN;

END.  <<procinfo example>>


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

If the previous program was executed by pin 45 in the process tree of figure 2, the following information would be returned:

```
item number  information
    1            45
    3            2
    4            5
    2            12
    5            3
```

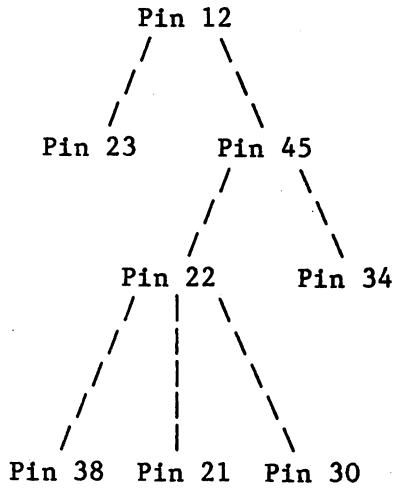


Figure 2

****

V. CONTROLLING PROCESSES

Once a program has created a process, it can control its activity. As mentioned before, it can activate its sons via the intrinsic ACTIVATE. However, only a father can activate a newly created process. ACTIVATE is called with the following parameters:

```
                IV    IV        O-V
ACTIVATE (pin, susp).
```

The process' pin number is required, but the susp parameter is not. If susp is provided and not equal to zero, then the calling process will be suspended and the specified process will be activated. Otherwise, the father process continues to run and the activated process becomes ready to run. The activated process will execute when the dispatcher selects it as the highest priority process to launch.

A process may also suspend itself. Via the intrinsic SUSPEND, a process may place itself in a wait state and state its expected origin of activation. The intrinsic calling sequence is:

```
                   LV   IV          O-V
SUSPEND (susp,rin).
```

The RIN parameter is the Resource Identification Number that will be locked for the process until it suspends again. The RIN allows a process to have exclusive access to a particular resource at a particular time. This is one way to synchronize processes and their resources running under the same job.

One other process control intrinsic is GETPRIORITY. When a process is created, it is given the same priority as its father. This intrinsic allows a program to change its own process' priority or that of a son. The process cannot, however, request a priority outside of its allowable priority class. GETPRIORITY is called as follows:

```
                    IV       LV       IV       O-V
GETPRIORITY(pin,priorityclass,rank).
```

The priorityclass parameter is a 16-bit word that contains two ASCII characters. Depending on the priority queue desired, the parameter is "AS," "BS," "CS," "DS," or "ES." (If a user has privileged mode, he can supply an absolute number for the priority parameter instead of the ASCII characters. It is done by supplying the parameter "xA where "x" is an integer value and "A" is the ASCII character A.) The rank parameter, once again, is not used except for compatibility with old versions of MPE.

The last two intrinsics to be discussed are used for process termination. When a process is terminated, it must return all the system resources that it is holding, stop its sons from running and start their termination sequence, and then request that its father take away its stack. The two intrinsics used for termination are:

```
      IV
KILL(pin)      and        TERMINATE.
```

The parameter in the KILL intrinsic is the pin number of the process' son that it wants deleted. TERMINATE can only be used for the calling process.

The following is another example using these various intrinsics. This example illustrates the CREATE, ACTIVATE, GETPRIORITY and TERMINATE intrinsics:

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

BEGIN

ARRAY NAME(0:15) := "EXAMPLE.PUB.SYS ";
BYTE ARRAY BNAME (*) = NAME;
INTEGER PIN;

INTRINSIC CREATE,ACTIVATE,TERMINATE,GETPRIORITY;

CREATE(BNAME,,PIN,,1);          <<create the new process. reactivate >>
                                <<the father when this one finishes.>>

IF <> THEN TERMINATE;           <<kill process because of error in>>
                                <<creation sequence              >>

ACTIVATE(PIN,2);                <<activate process and then reactivate>>
                                <<calling process by the son         >>

IF <> THEN TERMINATE;           <<process not activated due to error>>

GETPRIORITY(PIN,"DS");          <<change priority of son process>>

IF <> THEN TERMINATE;           <<new priority not granted>>

END.

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

VI. SUMMARY

This paper has summarized various intrinsics that can be used to create new processes, obtain information about them, control them, and then terminate them. A new intrinsic, PROCINFO, was also introduced which can provide the user with more information about processes without requiring privileged mode capability. MPE is a process oriented operating system, and with a better knowledge and understanding of how processes operate, a user can enhance his applications and their performance on the HP3000.

Putting the HP3000 to Work For Programmers

Thomas L. Fraser
Forest Computer Incorporated
East Lansing, MI

I. THE OPPORTUNITY

The demand for software is exploding as businesses and other organizations which use computers strive to be more productive, control costs, and improve the quality of management information. The acceleration of this demand is forecasted to continue throughout the early 1980s.

Software is produced for the most part by people, skilled people. These "programmers" are a limited resource. If the increasing demand is to be met, either the size of this resource must be increased or the productivity of the resource must be improved.

Looking at the issue from the viewpoint of an individual DP shop, increasing the size of the resource means hiring people. Skilled people are expensive, and costs are going up. Especially expensive are programmers, due to the already existing shortage. This shortage also makes it difficult to find quality people. So increasing the size of the resource is not always easy and is very costly.

Another trend that is evident is the decreasing cost of computer hardware. This contributes to the increasing demand for software, and thus is part of the problem. However, it can be made part of the solution by putting computers to work for the programmers.

This is the opportunity. Use the computer to increase the productivity of programmers. Provide software tools which allow the people to work efficiently and quickly. The expensive and scarce programmer should not have to wait for or adapt to the increasingly inexpensive computer. In a word, the computer needs to be made more friendly toward the programmer.

II. THE HYPOTHESIS

In the specific environment of the HP3000, programming is usually done online. A majority of the programs are written in COBOL with FORTRAN also popular. There are several types of tools which can be introduced to this environment. Report generators, high level file systems, COBOL generators, forms generators, and very high level languages such as RAPID/3000 can all help. However, in most shops programmers still spend a large amount of time at a terminal working with source code. This therefore is the first place to look when considering how to get the HP3000 working for the programmer.

By far the most prevalent software tool used by programmers is the HP editor. Compared to the primeval batch methods of source input and maintenance, EDIT/3000 is vastly superior. Because the editor is interactive, changes can be viewed as they are made within the context of the rest of the program. Also the editor provides many features such as searches and global changes previously unavailable. And best of all there is no problem keeping card decks in sequence.

However, the new features and capabilities come with a price, that of increased demand on the system resources. The programmers are competing with each other, as well as with production users, for precious disk accesses and CPU time. An obvious result of any delay in system response is lower productivity. This applies to all users of the system, including programmers.

EDIT/3000 is not without weaknesses. It is a line-by-line editor. This is a logical carryover from the days of cards. (Remember, the VDT was originally intended as a keypunch replacement.) All I/O is organized around the line as a standard unit. I/O from the terminal interrupts the hardware once for each character because of lack of block-mode handling. Moreover, the software must get involved each time "RETURN" is hit; this is a minimum of once per line with the exception of the "CHANGE" command, and with many commands can be several times per line. Disk I/O is blocked, but the binary search used to locate the card-image formatted records i very expensive in terms of disk accesses. This line orientation has obvious negative performance implications. Moreover, it means that the programmer must work with a line at a time. Despite the ability to display 20 lines on a single CRT screen, only one line at best can be entered or changed per transmission except for the noted exception.

Believing that the overall demand on resources might be reduced, and system performance improved, there still remain other areas to be investigated when seeking to improve upon the editor. For example, the "TEXT" and "KEEP" commands are very slow due to the fact they are actually file copying commands.

One of the nice features of EDIT/3000, that of being able to see the changes in context, is mitigated against by two major factors. The first is screen clutter. Unless one repeatedly does "LIST" commands, the screen becomes full of old source lines and already executed commands as well as current source lines. The second is the inability to access everything on the screen.

Performing some operations, even on a single source line, require several commands to be transmitted. This makes more effort by the programmer necessary, and slows the coding process. This, together with the other factors, are seriously impairing the speed of software development and system performance.

Thus the hypothesis, that a full screen block-mode editor, written for maximum features with minimum demand on machine resources, would dramatically improve programmer productivity. Improved response time for other users could also be anticipated.

III. THE METHOD

To test the hypothesis a full screen, block-mode editor was designed and written. The result of this effort, called "CHICKEN" by its architects, was a COBOL and SPL program which can be used to edit source code, documentation, stream-files, and other text. No operating system modifications are required, and the program runs in ordinary session-mode.

Block-mode transmissions dramatically reduce the overhead of terminal I/O. This is especially true when the line is driven from the Asynchronous Data Communications Controller (ADCC). The number of transmissions is also reduced making life easier for the programmer. The terminal has a microprocessor; block-mode enables taking advantage of this to reduce load on the HP3000. If one has paid for a "smart terminal," it behooves one to use it. By the way, CHICKEN can automatically switch the terminal between block-mode and character as needed. Implicit is that the VDT being used is an HP compatible terminal with block-mode capability.

Full screen access is another way of putting the terminal to work. With the new editor, all twenty-four lines of the screen are used. One line is for entering commands, one line for error messages, and the other twenty-two are used to display source lines. The programmer can change, delete, or insert lines of code any place on the screen by using just the terminal capabilities. Only after completing an entire screen, is the source transmitted to the HP3000. At that time CHICKEN will determine which lines should be deleted, changed, or added to the file. There is no need to use commands to tell it what is a change, delete, etc.

The disk organization of source files also effects significant advantages. Standard MPE files are used, but CHICKEN has its own access techniques. The old card-image format is replaced by a compressed format which is designed to maximize performance while using less disk space. Because of the file organization and access methods, CHICKEN can retrieve any single line of source code in one disk access, and any twenty-two consecutive lines in an average of 1.4 seeks with a maximum of two required. This single technique has great performance implications.

Ease of use is always an important design consideration and CHICKEN is easy to use. The command set uses language similar to EDIT/3000 to make it easy to quickly get acquainted. Any command can be issued at any time. Moreover, it is seldom necessary to issue multiple commands to accomplish a single task. Recall also, that the software frequently will figure out what you want done without having to be specifically told. Probably the biggest factor in ease of use, though, is the full screen access. A simple list of commands is below.

CHICKEN has other features which contribute to improved productivity:

- Screens are automatically formatted for COBOL, FORTRAN or SPL source if desired.
- The programmer has access to most MPE commands from the editor.
- Compiles can be submitted without leaving the editor.
- Special passwords are put on source files.
- An optional log of changes provides a means of recovery and a means of "backing out" modifications. This also can be used to provide an audit trail.

Several commands are listed below to show general syntax and to compare their operation with the similar commands available in EDIT/3000. In general, the commands follow a standard format as shown here:

CMD <starting-line <ending-line>> required-params <optional-params>

CMD <starting-line <ending-line>> required-params <optional-params>

Most command key-words are the same as found in EDIT/3000, and all can be invoked by entering only the first letter. For instance, "LIST 120.5" can be entered as "L 120.5".

CHICKEN attempts to give the user as much flexibility in entering a command as possible, so as to accommodate differing user styles acquired through exposure to various other editors. Thus the following commands would all have the same effect if entered:

DELETE 20/30
D (20.00:30.00)
DEL 20 30
DELETE 20,30

The goal here is to make the editor easy to learn by not requiring strict adherence to particular syntax rules, and easy to remember by keeping command formats simple and regular.

Following are some representative commands:

TEXT edit-file < NEW < mpe-source-file > >

This command opens and grants access to an edit-file.

If another edit-file is currently open and being worked on, it is automatically closed. If the NEW option is entered, a new edit-file is created. The "mpe-source-file" refers to an EDIT/3000 source file which can be copied to the CHICKEN edit-file. This command executes very quickly because there is no copy operation from a source file to a work file as in EDIT/3000, except when an MPE source file is copied in, which happens only rarely.

KEEP < A < B > > mpe-source-file < PURGE >

This command makes a copy of the currently accessed edit-file to an EDIT/3000 formatted source file. Normally all lines will be copied. If line A is specified, all lines from line A through the end of the edit-file will be copied. If line B is specified, the copy will only include the lines from line A through line B. If the PURGE option is entered, the edit-file is closed and purged from the system after a successful copy operation.

This command is used infrequently, usually for backup purposes. Since compiles can be implemented directly from within the editor on the existing edit-files, there just isn't much need to KEEP files. If one edit-file is TEXTed in and modified, a second TEXT automatically closes the first edit-file with changes intact. The improvement in response time to access edit-files can be dramatic even on only a moderately loaded system.

LIST < { A / LAST } >

A simple LIST command without parameters will display the first 22 lines of text in the edit-file. Subsequent transmission will display the next 22 lines, in effect paging through the text. If line A is specified, then line A and the next 21 lines of text following line A will be displayed. Again, paging applies after entering the command once. If "LAST" is specified, then the last line of text and 21 blank lines are displayed.

This is where some of the power and flexibility of a full screen block-mode editor can be seen. The user can now be free to move the cursor anywhere on the screen, modifying, inserting, and deleting lines. Changes can be reviewed in context of the surrounding text. Even line numbers can be changed simply by typing over the old ones displayed. All of this goes on without bothering the host computer. Of course, this frees up the HP3000 for other tasks at hand.

FIND < A < B > > *textl* < ALL >

This command performs a search for the next occurrence of textl and displays the line containing the textl along with the following 21 lines of text. If line A is specified, the search will begin at line A and continue until a match is found or the end of the file is reached. If line B is specified, the search will only encompass lines A through B. The asterisks surrounding textl represent delimiters, which can be any non-alphanumeric characters including a space.

Examples:

FIND MEN	<spaces as delimiters>
F/ALL MEN/	<slashes as delimiters . . . space is part of the search string>

The ALL option will cause the editor to attempt to find all occurrences of textl and display all corresponding lines. If 22 occurrences are found before the search line limit, the lines containing occurrences of textl are displayed along with a message stating that the search is not finished. The user can modify any of the lines on the screen. To resume the search, the user only needs to enter "F", and the editor picks up the search where it left off. The user can even begin a search and then use other commands such as LIST or CHANGE, add and delete lines, etc., and will still be able to resume a search.

RENUMBER < A < B > > < BY N >

Renumbers the edit-file. If no parameters are entered, all text lines are renumbered. If line A is specified, the numbering will begin at line A and continue through the end of the file. If line B is specified, the renumbering will only be done on lines A through B. The BY N option allows the user to override the default line number increments used by CHICKEN in a renumber operation.

The renumbering is done to the file in place, rather than through a copy procedure. This significantly speeds up the operation in comparison to, say, EDIT/3000's GATHER command.

Other commands found in EDIT/3000 as well as many other line editors, are either unnecessary or have their utility reduced with a full screen editor. The ADD command in EDIT/3000 is a good example. With CHICKEN lines are inserted right on the screen between other lines of text, and transmitted back to the editor. The line number does not even have to be included. The editor identifies the surrounding text and calculates a line number for the new line. To add text at the end of a file, the user enters L LAST. CHICKEN displays the last line of text followed by 21 numbered blank text lines. Along the same vein, line deletes can be handled on the screen simply by placing the letter "D" before a displayed line of text. The DELETE command itself is only needed for global deletes, as in D 100/200.

The above are just a sampling of the full screen editor's command list. As was mentioned previously, command keywords have, for the most part, been kept the same to facilitate learning to use CHICKEN. Thus the user will find such familiar key-words as GATHER, JOIN, HOLD, CHANGE, EXIT, etc., along with a few new commands, such as ZIP which initiates a compile for an edit-file without having to either exit the editor or KEEP the source code.

THE RESULT
(A Personal Digression)

The most notable difference upon the installation of

CHICKEN was not programmer productivity, it was programmer euphoria. After using it for even a short time, one gets hooked. In our shop we have a mixture of block-mode and character-mode terminals used for program development. To say that the character-mode terminals are collecting dust would be an exaggeration, but we have noticed people arriving quite early in the morning to stake claim to a "CHICKEN" terminal.

The productivity, response time, and performance improvements are also accomplished. As of this writing (December 1981), quantitative data are not available. Anyone desiring more information of this nature, or having any further interest in learning more about CHICKEN can write to:

Tom Fraser
Forest Computer
P.O. Box 1010
East Lansing, Michigan 48823

or call (517) 332-7777.

RPG — A Sensible Alternative

Steve Wright

PREFACE

The main purpose of this talk is not to present evidence of one programming language being better or superior to another. The decision of primary language is most likely already been made in your operation. This talk, however, is designed to make the statement that RPG is being used successfully in the HP world and should not be ignored becuase of dominate usage of any other language. Also, I want to present some uses that you may not have considered. Please bear with me on some elementary topics. But, I feel that some users have not been exposed to them, and hopefully will be beneficial.

FACTS (AND HISTORY)

Consider the following:

- The most common programming language by far is not RPG.
- RPG to COBOL conversions are commonplace and packages that will perform 99% of the work are available from several vendors.
- Several former RPG users were told to switch to COBOL to get the most out of HP machines (mainly with support considerations).
- The person next to you at lunch today sneered at you when he heard that you used the RPG compiler.

WHY DO WE INSIST ON USING SUCH AN ANIMAL?

Consider the following:

- COBOl shops are experiencing low productivity levels and are seeing report writers as an escape route.
- Operations that escaped the IBM System 3 world with excellant track records are using HP equipment to enhance a thing that is already good.
- Managers are finding out that matching programming languages with the assigned task can be a rewarding adventure.

My own personnel experience with Hewlett Packard equipment and available programming talent has led to the following situation. I have 90% of my programs in RPG, 8% In COBOL and 2% In FORTRAN. I am using FORTRAN In heavy math oriented problems, COBOL for mass data entry programs and RPG for all batch and Quicky Del routines. The heavy terminal usage programs are In COBOL and FORTRAN, and therefore are the heaviest used. But the RPG programs that support edits, update and reporting are the real workhorses that support every system.

My reasons for using RPG in so many batch programs is the speed at which the programs can be written, tested and Implemented. The run time difference between RPG and COBOL in batch routines for the most part has not been substantial and the implementation schedule of entire systems can be speeded up. Batch is not a daily task for the most part and therefore can be paid for in machine cycles instead of programming dollars.

Most programming managers have grown to love compiler languages with the ability to go between machine lines with minimal difficulty. The report writers of today are very impressive. I have looked very carefully at one very good one, and I still may ask Santa Claus for one next year, but if I were a small shop with limited programming resources (salary dollars not talent), I would have to consider RPG for everything from batch to data entry programs.

HOW I DO IT

Below are some tricks of the trade that I use effectively to enhance my operation. Again, some of these items are elementary to many who will read this paper, but I find most people will find one or two things that will be new to them.

RPG supports only one-dimentionial arrays. I use some algorithms to make two-dimentionial arrays work. Suppose I wanted an array of 12 years data on the total of 6 product lines. To define the array, specify 72 entries (execution time array). To load the entries use X=year to load, Y=Product Code (1-6). The following code will locate the element.

```
*
*        X      SUB  1     X
*        X      MULT 12    X
*        X      ADD  Y     X
*    USE THE X ELEMENT OF THE ARRAY
*
* TO FIND THE MEANING OF THE ELEMENT SPECIFIED BY THE
* VARIABLE "WORK", USER THE FOLLOWING:
*
```

```
*  WORK   DIV  12   X  (NOT HALF ADJUST)
*  X      MULT 12   Y
*  X      ADD  1    X
*  WORK   SUB  Y    Y
*
*  THUS X= YEAR LOADED  Y= PRODUCT CODE
*
```

You may choose to print the results out using a 6 element array running down the loaded 72 element array by element and printing when you have filled up the 6 element array. I use just such a routine to look at the past 10 years history for product trends.

R.A.F. (Random Access Files) (Addrout)

HP has released "Xsort" that will sort only using the key fields and the relative record number and dropping the large data portions of the record and leaving a file that contains only the relative record number of the records in the file arrainged in the order the file would be in if the entire record was carried along. The scheme allows very fast sorting of very large data files. This discussion is fully explained in the communicator #26. Use it. It can be a life saver and a hero maker. In case this all sounds familiar, It is the System 3 Addrout processing. Also, in processing files by random access (no keys) do not be afraid to declare a MPE file file as input chained and simply read it by relative record number. I do it all of the time to position myself back and forth within the file. "Chained" does not always mean "keyed."

DSPLY

When using the DSPLY command, the limit of 8 characters for a constant in factor one is annoying. I use a compile time array at the end of the program to detail my prompts, giving me all of the characters I need without many cumbersome moves. Also keep in mind that you can send escape sequences in the DSPLY command to manipulate the terminal as well as ring a few bells. Very fast data entry programs can be written using DSPLY. I wrote one data entry program using DSPLY in two hours thinking that it was going to be a one-time program. It is still in use two years later with no changes.

SETLL

The "SETLL" command is very useful in either KSAM or IMAGE file. Everywhere you have a user screaming for a name search routine, use the name as a duplicate key (or automatic data set) and use processing limit (SETLL) and the read command to give amazing results. The following code is an example:

```
*             ASKNM     TAG
*            'NAME?'    DSPLY TERM       NAME
*            NAME       SETLLMASTER
*            LOOPER     TAG
*                       READ MASTER              99
*       99               GOTO ASKNM
*            FULLNM     DSPLYTERM
*            'CONTINUE'DSPLYTERM         ANSWER
*            ANSWER     COMP 'Y'                 98
*        98             GOTO LOOPER
*              'STOPPED'DSPLYTERM
*                       GOTO ASKNM
*
```

"LET ME CALL YOU SPL — (OR COBOL OR FORTRAN)"

The HP RPG compiler allows exits (calls) to external routines that can written in other languages. If you feel this is needed, keep in mind that it is available. The Orlando swap tape has such routines (such as calls to system intrinsics) can be very useful.

INTERACTIVE WITHIN THE PRODUCT LINE

RPG and V/3000 is not a bad combination. The main reason I used COBOL with DEL was the ability of COBOL to read only one field at a time (a real time saver). However, the main thrust of V/3000 is in reading the entire screen at once and allow the V/3000 routines handle screen painting and repainting. (It does a pretty slick job at times.) RPG looks like a natural for usage with VIEW. V/3000 uses more screen dependent controls that insulate the programmer from the messy calls, that one should try it in RPG.

"HOW ABOUT A DATE GOOD LOOKING?"

Communicator #24 tells of how the RPG programmer can specify "F" in column 17 of the header spec to

allow him/her to specify the sysdate from other sources than the system clock. One may use a disc file, or request the user enter the date as he runs the program. Also, the time2 command is explained in the #24 communicator. This command will return to the user the date in the format of "THU, JAN 10, 1980 9:25 AM JULIAN:010." This can be very useful in that the compiler allows you to select which fields in the above format you want to see, thus reducing the moves that would normally be associated with it.

"CHECK PLEASE" (MOVEA)

I have a rather classy check protection routine that I am contributing to the swap tape. Use it in good health. The routine uses "MOVEA" extensively. The "MOVEA" usage is worth going through with beginner programmer types and intermediate types who have not been exposed to it. The full usage of arrays can enhance their productivity.

CIRCLES AND CYCLES

The RPG programmer whether beginner or advanced, must, must, must, must know the RPG cycle. If he/she is not taught early, you simply have a COBOL or Assembler programmer with very restricted limits. The main attraction of the language is having the cycle do all of the grunt work for you. If you still think that total time processing occurs after a break instead of before a break, (there is a differance) you need to spend some serious time with the cycle flowchart in your HP or IBM Reference Manual.

"WHERE DO WE GO FROM HERE??"

RPG programmers are a sturdy lot, but with the advantages of an international user group at our disposal, we should be doing more in the area of sharing ideas. At the San Jose meeting, several RPG users wanted to get together and start a special interest group, or a newsletter. The interest was high but no one kept the momentum going. I am open for suggestions for ways to start such a group. Some suggestions I have heard for the users are as follows:

- Have a newsletter with shared ideas as the emphasis
- Start a special interest group for addressing HP
- Set up a network of "help-lines" for RPG users
- Have special user group meeting before or after the international user group meeting each year.

One other item that I think may help is using some of the system 3, 34, 38 aids. That includes getting free subscriptions to such publications as *Small Systems World*. It is a monthly publication that has been well accepted by many small systems users. If you want the address to ask for the free subscription, please contact me.

Techniques for Testing On-Line Interactive Programs

Kim D. Leeper
Wick Hill Associates Ltd.
Kirkland, Washington

ABSTRACT

This paper will describe various strategies for testing on-line interactive programs. These strategies include acceptance/functional testing, regression testing and contention testing. The paper will also discuss the mechanics of testing including testing by human intervention and various forms of automated testing. This information will allow you to create a viable test plan for software quality assurance in your shop.

INTRODUCTION

Program Testing. Those two words undoubtedly conjure up thoughts of long boring hours sittig in front of a terminal typing in all kinds of data looking at error messages produced by the program. This paper will present alternatives to this type of program testing. It will also describe a prototype test plan or quality assurance cycle which may provide the reader with ideas for implementing his/her own test plan for his/her own shop.

We must make sure we are all talking the same language so some definitions are in order at this point.

What is Testing?

Software testing may be thought of as a series of data items which when presented to the program under test (PUT) cause the software in question to react in a prescribed or expected fashion within its intended environment. The purpose of testing is to expose the existence of mistakes in the program or to show the absence of any such bugs. If the software does not act in the expected way then one has found a bug or mistake in the program.

Vocabulary

SCRIPT — a list of inputs or data items given to the PUT for testing purposes.

DATA CONTEX OF BUG — the collection of inputs required to cause the PUT to fail or return results which are not expected.

TYPES OF TESTING

Acceptance/Functional Testing

This type of testing is used to demonstrate that the various functions of a given software package actually works as described in its documentation. This is not exhaustive testing as it only examines one or two transactions per function. This is the typical type of testing the vast majority of users perform now.

Regression Testing

This type of testing can be used to test all the various logical paths within a given software system. Regression testing tries all the data extremes per function that the program could be expected to respond to. This type of testing is rarely performed because it is resource, that is to say hardware and personnel, intensive.

Contention Testing

This type of testing is used to determine if the database or file locking strategies that are used in your application programs actually work. Two programs are executed at the same time, one performs a transaction which locks a given item in the database. The second program attempts to access this same data that is secured by the lock via another transaction type different than the one used in the first terminal. The designer in this instance is interested in the message of action of the software to this challenge. This type of testing becomes particularly relevant when the installation has many programmers implementing many systems dealing with the same database.

THE TEST PLAN OR QUALITY ASSURANCE CYCLE

The keystone of any successful testing program is to have a viable test plan. This plan should describe all the phases a software development project goes through and then ties all the phases together in one comprehensive flow of data and actions. The plan should extensively use feedback loops so that when problems are discovered there are clear paths for the problem rectification process to follow. One possible quality assurance cycle that can be proposed may be seen in Figure 1.

The diagram indicates that the test script should be generated along with the design of the software. Many times in the design process the designer realizes some weakness in the design and will want to specify a special test in the scriptfile. S/he is encouraged to do so. Many companies that use this methodology specify programs

by a test script and V/3000 screens.

Examining this diagram more closely one can see that the flow of debugging actions is closely tied to the design/maintenance of the original test script. The reason for this is to force the implementors to keep track of the bugs they discover and place them in the test script. This script should then be run against the application program whenever a new fix or correction has been applied to the original program. This script will constantly force the program to re-execute all the previous transactions which caused bugs to occur in the past, to assure the program maintenance team that no additional mistakes have been introduced by fixing the last bug.

In this version of the QA cycle the users are always in a mode of testing the delivered software. Eventually the users will find a bug which will start the whole cyclic process over again. If they don't find a bug, don't think it is not for trying. The users have eight hours per day per person to find bugs. It does not take very long before they have more execution time on the application software than the designer/implementator has. This is the time when more bugs can and will be found which will start the cycle once again.

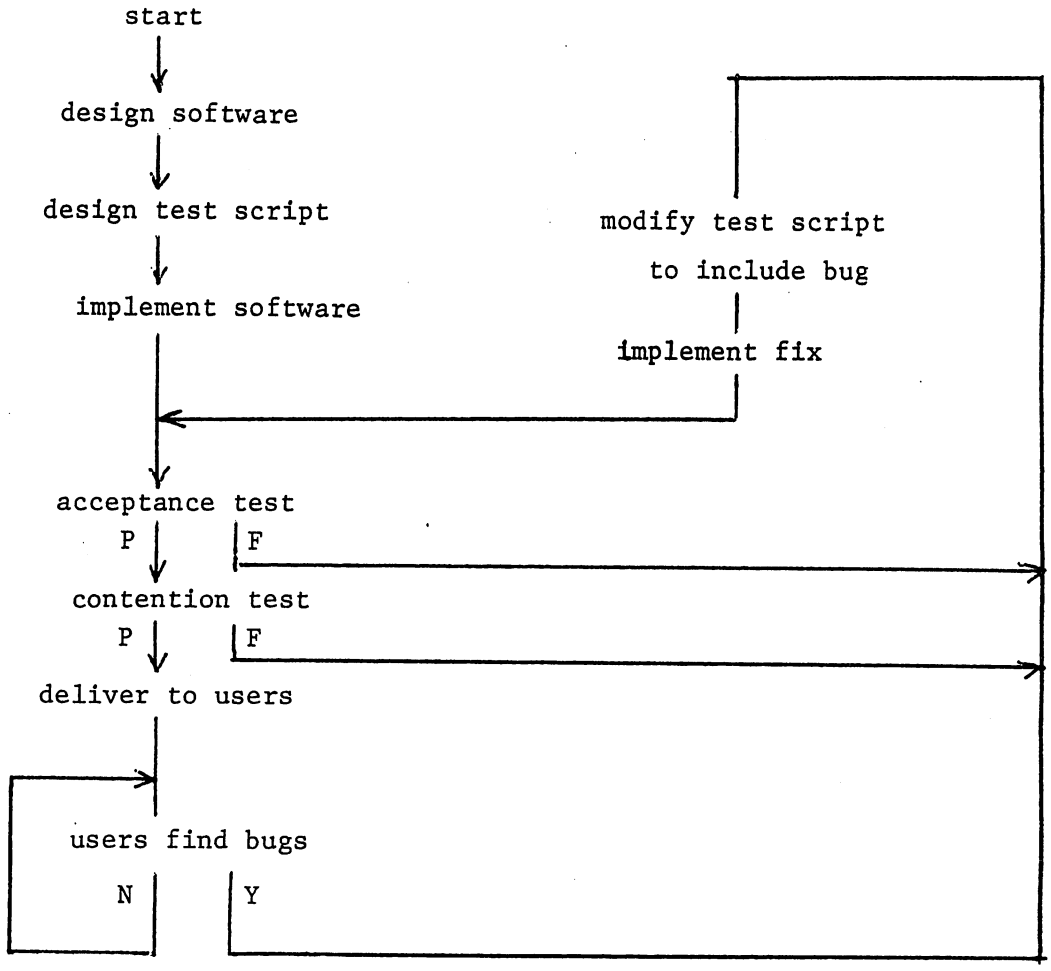


Figure 1
Quality Assurance Software Cycle

THE MECHANICS OF TESTING

Obviously, the type of testing that is currently being used is human intervention testing. This is where a programmer of analyst sits in front of a terminal and simulates a user by following a handwritten script. This approach to testing is less than desirable for a number of reasons, among those being:

1. input data error due to arrogance/boredom in applications tester;
2. non-repeatability of exact timing due to human tester;
3. the tester might not record everything happening off the screen;
4. an expensive employee is being utilized for testing purposes when s/he could be designing/implementing more applications

A possible solution to the dilemma outlined above is to mechanially examine the software by exhaustively

testing all the paths in the program by computer. Using completely random data types as input you could automate the testing process. However, as there is only so much time available during a 24 hour day it might take all day to exhaustively test a very small application program. This technique is machine bound in terms of both creating the random data and testing all the paths in the application code.

A saner approach would be to combine the above two techniques into a testing procedure that utilizes a human being's capacity for creative thought and a machine's capacity for highly efficient repetition. This technique would rest in the programmers designing the scripts used for automated testing at the same time as they design the application itself. Once the test script is produced then the machine itself tests out the application program under the watchful eye of a human. In fact the script can be used as a specification for implementing the system. As Yourdon has written, "What we are interested in is the minimum volume of test data that will adequately exercise our program."[1]

It is now possible, using VTEST/3000, to automate this testing procedure and achieve a real manner of quality control. VTEST/3000 includes full V/3000 testing capability. The compiled code runs as though it were in a live situation with VTEST/3000 providing full documentation of all errors occurring on the screen of the terminal.

In order to use VTEST effectively one must appreciate the diagram in Figure 2. There are two types of tests that VTEST can perform, block mode testing for those programs that use V/3000 and non-block mode testing for those not using V.

The first type of testing that will be discussed is non-block mode application testing. In this case VTEST looks like a non-block mode glass TTY terminal. The script file contains the actual commands and data that a user would normally type into the screen of a real terminal, everything between and including HELLO and BYE. This script file is built and maintained by the standard HP EDITOR. The script file is input to VTEST. VTEST transmits this file a line at a time to the application and VTEST prints out a report of the terminal screen before the return key was depressed and after along with the number of seconds that the response took to come back to VTEST.

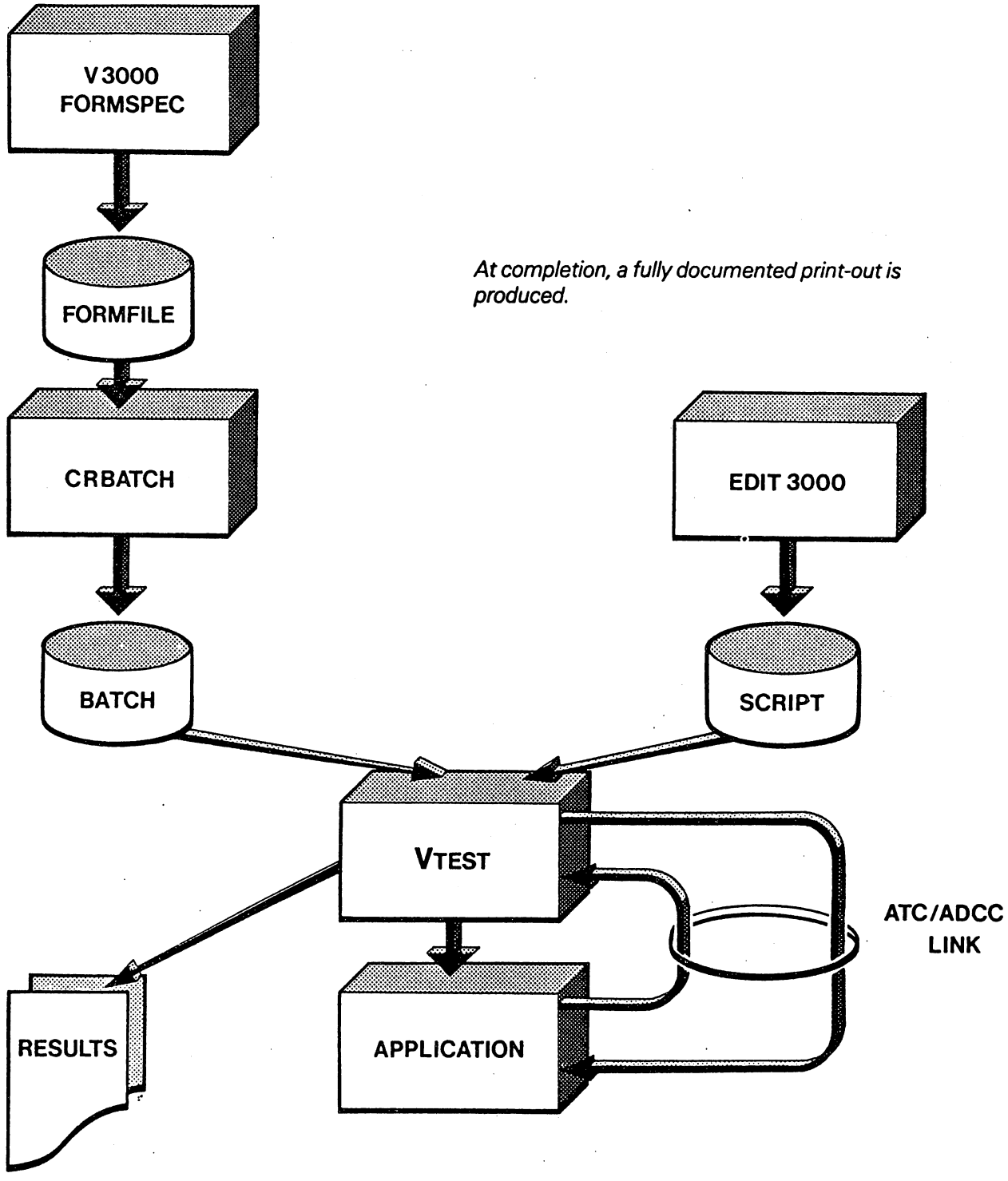
The second type of testing that will be discussed is block mode application testing. In this case VTEST looks like a HP2645 block mode terminal. The script file is the same as above with an important extension. The script file now can tell VTEST when it must transmit data to a V screen. The data for a V screen must come from a different type of file. This file is called the BATCH file. This BATCH file is created and maintained by another program called CRBATCH. CRBATCH allows the user to specify the formfile name and the form to be displayed. Data is then entered and CRBATCH reads the screen and puts the data into a BATCH file. CRBATCH allows the user to insert screens, to delete screens and modify the data in screens already in the BATCH file. It is a general purpose maintenance program or editor for BATCH files. Whenever the application program under test wants some block mode data the next record is read from the BATCH file. VTEST then transmits this record complete with all the special characters that V requires to the application. VTEST prints out a report for every transaction before the ENTER key was depressed and after the next screen was received along with the number of seconds that the response took to come back to VTEST.

One can see quite easily that VTEST fits right into a well designed quality assurance cycle.

REFERENCES

[1]Edward Yourdon, "Techniques of Program Structure and Design," Prentice-Hall, 1975.

[2]Software Research Associates, "Testing Techniques Newsletter," (415) 957-1441.



At completion, a fully documented print-out is produced.

Figure 2

A Universal Approach an an Alternative to Conventional Programming

Bill McAfee and Craig Winters
Futura Systems
Austin

Some two years ago we set about to find a shortcut to programming, a way to simplify and speedup the actual coding, to eliminate all or nearly all of the housekeeping, and to improve the reliability and maintain-ability of our work. We wanted to be able to deal with any problem in terms of the logical operations to solve it, rather than with a sequence of detailed programming statements.

We identified approximately 100 routines to handle input, validation, conversion, formatting, and other functions not provided in the System Library. We designed an English-like language and compiler to invoke these operations as well as those in the SL and to pass them parameters; and we designed a driver to execute all operations in a reliable, consistent manner.

Our primary objectives were:

- to define data types by the significance of their contents (date, phone, zip code, quantities, monetary amounts, etc.) and to perform data entry, validation, conversion and formatting automatically, regardless of storage type.
- to provide a very high-level English-like language that would be both easy to learn and self-documenting.
- to be able to use any number and type of files simultaneously including multiple databases, datasets, KSAM, printer, etc.
- to automatically store and load tables to supply values needed at run-time.
- to simplify declarations and eliminate the dull, boring redundant part of programming, where most errors are made
- to provide text specification syntax, including literal text, program variables and control characters, for use as program messages, report output, headings, etc.
- to work equally well for interactive and batch applications.

We wrote the system in SPL. It has been in daily operation for just over two years, during which time there have been two major rewrites and many additions and enhancements designed to further simplify its use and improve performance. Presently we are just putting the finishing touches on the final version which will incorporate all the things we have learned from these past two years of use and will reflect at every step what we feel will be the best design and coding available.

Since this is a new and unique approach to programming, there is no generic for it. We call it The Futura System, and it consists of a language, compiler, driver and an extensive procedure library. We have attempted to give it the ability to do anything, and when we have discovered something it would not do, we have added it. And while our primary intent was to use it for applications programming, we have found that it is equally strong and valuable as a powerful, versatile utility that is able to supplement and round-out the various system utilities quite handily.

Programming using FUTURA consists of Initialization Commands and Mainline Commands. The compiler reads and validates these, checks their parameters, provides default values where desirable, builds the Mainline binary command module, and formats and prints a program listing in one of several styles. The binary command module resides in the data stack and drives and controls the entire program execution.

InitCommands include:

STACK — which sets the total space the program will require. It has a default value of 3500 bytes, which will handle most utility needs as well as quite a lot of applications.

ALLOCATE — which dimensions the various buffers, should the defaults not be quite right.

BASE — used to open an IMAGE database.

SETS — for identifying the DataSets to be accessed.

FILE — for opening MPE and KSAM files.

TABLE — declares and loads a table, taking care of data conversion, statistics and storage automatically.

PRINTER — opens a printer file according to your specifications, including headlines, page numbers and location, forms-message, and all other parameters used with the line printer.

LOAD — which initializes any area in the data stack with any string or binary value.

INTEGERS — used to load a string of binary single-word integers at any location.

IDENTIFY — an InitCmmd that may be used or implied by the syntax, it sets up a table of identifiers for use throughout your program.

All InitCmmds that may be required must precede the Mainline.

Mainline Commands are names of logical operations such as ADD, MOVE, UPDATE(datatype), BINARY, etc. They may have up to five parameters, some of which are required and some optional. There are MainCmmds to do everything, and frequently there are several, giving the programmer meaningful options on how to accomplish a step. For interactive applications there are a dozen-or-so UPDATE (datatype) commands, such as UPDALFA, UPDNMBR, UPDZIP, UPDSSNO, etc., which not only accepts, validates, formats and displays, and stores the data, but also gives the programmer complete control and recognizes up to 8 special characters that permit backing up one or more fields, begin record over, check for mail, etc.

The fact that commands are the names of logical operations rather than language requirements means that when you have logically solved the problem you have also largely written the program.

Many MainCmmds return one or more values to the program such as the Condition Code, Length, DBStatus, Returned Value, etc. as may be needed.

Text strings to be used as prompts for interactive operations are passed automatically to the program, as the compiler counts them and stores them together with any control characters needed to handle the screen and make an eye-appealing presentation. Text needed for any other purpose is also passed, counted, stored, and recalled with little or no effort on the part of the programmer.

The MainCmmds themselves, the Identifiers, and the way the text strings are handled all provide a great deal of self-documentation right where it is needed in a program, and other documentation and comments may be added at any point. There is an index-building facility that produces an index for the documentation consisting of the program name and all of the comments in each program.

Many commands provide for testing and branching. They are processed uniformly by a subroutine, and branching may be either to a label or to another instruction. Subroutines may be nested up to 20 deep; they may call themselves, and they may reside anywhere in the Mainline. There are both Init and Mainline $INCLUDE commands, allowing routines to be stored separately where they may be used by several programs by including only the reference table.

The binary module together with any tables and initial values may be automatically saved and used again without recompiling by simply adding "$" to the STACK command ($STACK). This binary file may be purged any time changes are made, and it will be recompiled and saved at next compile if the "$" is in place.

In the handout pages we have included examples showing the program file as it is keyed using EDITOR, the normal program listing provided by the compiler which formats this Editor file and prints the permanent documentation, and a look as the terminal screen as each of these programs would appear when run, and a sample of the printer output where applicable.

These are some of the programs that were used to produce the Proceedings and the Exhibit and Conference Guide. While we asked that the papers for the Proceedings be keyed in cap and lower case using the EDITOR, with standard 72-byte records, the facts are that everyone used his/her own method — with record lengths from 60 to 160 bytes and some embedded control characters that would completely snarl our typesetting computer if they were not removed.

These are mostly small, simple programs that will illustrate the truly universal nature of the Futura System as a powerful and versatile utility. I have also brought the documentation for the Automatic TimeSharing Accounting and Billing System (ATSABS) which will show how it can be used for a large, complex system.

This will also show the automatic indexing and system documentation features that are available. We would be glad to have you all look this over and discuss it either at our booth or at other times and places by arrangement. This system totally automates our TimeSharing accounting and billing. It required approximately 5,000 lines of FUTURA code, and we estimate it would have required more than 30,000 of SPL.

The Technology of the QUAD Editor, Part II

Jim Kramer
Hewlett-Packard
St. Louis, Missouri

INTRODUCTION

The QUAD editor is a text editor that was contributed to the Users Group library last year at the Orlando Users Group meeting. It has several features which make it notable and useful, the most important of which are that it texts files instantaneously and that it can undo any or all editing changes. A paper in the proceedings of that meeting, titled "The Technology of the QUAD Editor," described the implementation of these features.

In the past year QUAD has acquired many new capabilities, including the ability to maintain multiple versions of a file, to cancel the effects of the preceding command, and to compile programs. The purpose of this paper is to describe the implementation of these new capabilities.

A BRIEF DESCRIPTION

QUAD is a line-oriented editor similar to EDIT/3000 and TDP/3000. Its most important capability is instantaneous texting, and sometimes instantaneous keeping of files. A file is texted just by opening it and changes are kept in a separate work file. If the only changes to a file are modifications of existing lines, then keeping is done by posting the changes back to the texted file. Since changes are kept in a separate work file it is easy to undo any or all changes just by removing them from the work file: QUAD's UNDO command does this.

It is important that QUAD be able to find lines in the texted file quickly. QUAD starts out with no knowledge of the location of lines in the file, and must find requested lines using binary search. However, QUAD keeps a record of all blocks read during the search process and uses this record to shorten subsequent searches. The method is described in a paper titled "A New Tool for Keyed File Access (Sometimes)" in the proceedings of the Users Group's 1980 North American meeting in San Jose.

Features that are new to QUAD in the past year include the following:

1. Operating directly on changes to the work file, by means of the MODS key word in a line range. Changes can be listed, kept, and otherwise operated on. For example, "List Mods" lists all changes that have been made to the current file, and "Keep Modfile(Mods)" saves the modifications in a file named Modfile.
2. Cancelling the most recent command which modified the file. The Cancel command does this.
3. Maintaining multiple versions of the file being edited. The Freeze command prohibits further changes to the current version and starts a new current version. Prior versions can be read at any time, but not modified, by using the VERSION keyword. For example, "List Version 1" lists the first version, and "Keep Filename(Version 1)" keepts it.

TICKET FILES

The important characteristics of QUAD work files — variable length keys and data and re-use of space — are provided by a file access method which I call ticket files.

With most file access methods, the user who wants data stored specifies where it is to be stored — a record number. With ticket files the user does not specify; instead he just supplies the data to the access method and receives back a "ticket" telling him where the data has been stored. In order to retrieve the data at a later time, he must supply the ticket.

It is important to recognize that this technique gives enormous flexibility to the file access manager. The data can be put in the most convenient spot, for example a block that is already in a buffer in main memory. Within the block the record can be placed wherever there is space. With ticket files a record need not even be placed contiguously within the block — it can be broken into pieces.

Ticket files turn out to be perfectly suited to those applications in which data is found through pointers: tickets are really just pointers.

In order to make ticket files satisfactory as work files, it was necessary to implement a keyed sequential access method based on ticket files. The implementation is significantly different from KSAM and actually more powerful: both keys and data can be variable length, space is re-used, and keyed sequential access can be either forward or backward.

When a key is stored, a ticket is stored with it. The

ticket points to data. Thus storing data by key is a two-step process:

1. Store the data and receive a ticket.
2. Store the key and the ticket.

Retrieving data by key reverses the two steps:

1. Supply the key and receive the associated ticket.
2. Use the ticket to retrieve the associated data.

THE WORK FILE BEFORE MULTIPLE VERSIONS

Before describing how the current QUAD maintains multiple versions of a file, I will describe how earlier versions maintain the work file.

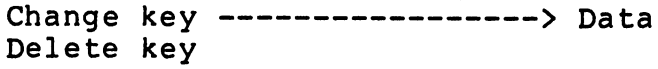
A ticket file is used as a work file, and contains two types of keys within the key structure: keys describing deleted ranges, and keys describing new or changed data.

To do a deletion, QUAD makes a single entry in its work file which is just a 17 character key. The first character is a "D" (for delete), the next 8 characters are the lower line number in the range, and the last 8 are the upper line number.

Since deletion is achieved with a single work file entry, it is very fast, and the speed is independent of the number of lines being deleted.

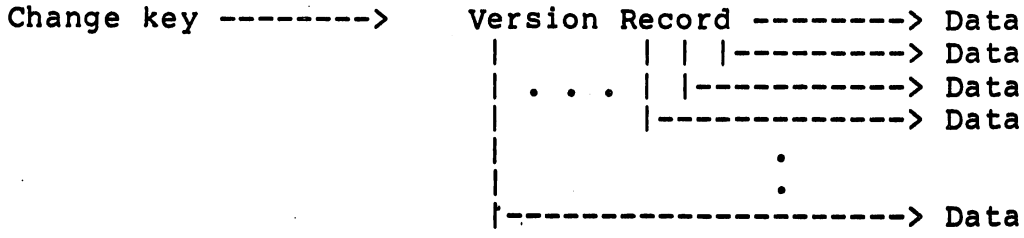
A change entry consists of both a key and data. The key is just the letter "C" followed by the 8 character line number, and the data is the line of text corresponding to that line number.

Schematically this structure is as follows, with an arrow representing a ticket held with the key and pointing to the data.



IMPLEMENTING MULTIPLE VERSIONS

Multiple versions were implemented by introducing a version record for each key, as follows:



Again the arrows represent tickets. In this case a ticket is stored with the key and points to a record called the version record, which itself contains one or more tickets pointing to data. With each such ticket there is a number identifying the version to which the data belongs.

Using this structure QUAD can, for each line number (represented by the change key in the figure), maintain multiple versions of each line.

Version records are themselves variable length, each being large enough to hold tickets for all versions of the corresponding line. There are generally many fewer versions of a particular line than there are file versions, because a given line will not change with each version of the file. A version record is restricted by QUAD internal buffering to 31 versions, but in general this will allow hundreds of file versions.

To operate on a version of the file, QUAD must integrate all deletions and changes for that version and all previous versions with the originally texted file. The algorithm to accomplish this is one of the most difficult I have had to write, and is complicated by trying to optimize performance. One performance problem that arises is that modifications to early versions which have since been deleted can slow down access to later versions.

IMPLEMENTING THE CANCEL COMMAND

The Cancel command cancels all changes made by the most recent command to change the file. Two consecutive Cancels have no net effect: the second cancels the effects of the first.

The Cancel command was almost trivial to implement once multiple versions had been implemented. The technique used was to reserve space in each version record to save a version number and ticket. Then when a command changes a line, the previous version number and ticket can be saved in this space. The Cancel command then just restores the saved version number and ticket to its prior place.

The only other implementation requirement for the Cancel command was to link together all the changed version records. This was easily accomplished using the tickets of the version records.

COMPILING FROM WITHIN QUAD

QUAD allows compiling for five languages: COBOL, FORTRAN, RPG, SPL and PASCAL. The syntax of the commands for compiling is identical to the syntax for the corresponding MPE command. However to compile from the file currently being edited, it is necessary to replace the text file part of the command with a line range enclosed in quotes. For example:

```
/SPL (ALL),$NEWPASS
/FORTRAN (20/40),USL,*LP
```

The ability to compile from the file being edited turns out to be especially useful for FORTRAN, because it permits compilation of single subroutines.

All compilations are done by invoking the requested compiler as a son process. File equations are set up for all specified files, and the compiler is passed a PARM to tell it which files were specified.

Whenever a line range is being compiled, QUAD passes the line range to the compiler through a message file. Message files are a new file type for MPE as of MPE IV.

This was my first experience with message files, and I encountered the following problems:

1. Unless the message file is built to contain only a single block, all blocks are posted to disc. QUAD uses a single block message file to prevent this posting.
2. If QUAD fills the message file before the compiler opens the file, QUAD's next write will fail with an end-of-file error. In this case, QUAD must loop, pausing and trying to write until the compiler gets the file open. Once this occurs QUAD will automatically be suspended by the file system on trying towrite to a full file, as long as the compiler has the file open.
3. QUAD must be careful not to send a null file (no records) to the compiler, because the file system will suspend the compiler indefinitely on its first attempt to read a record regardless of whether the file has any writers.

CONCLUSION

QUAD was created to quickly list files and make simple changes. I believe it or a similar tool belong in every 3000 shop as a significant resource saver.

A few users now use QUAD rather than EDIT/3000. This pleases me because I think QUAD deserves it, although there are still things that EDIT can do which QUAD cannot. However, I suspect that most users of these tools would quickly abandon them for general editing were a good full-screen editor to appear. I know I would.

If there is any permanent significance to QUAD, I believe it is to be found in the ticket file access method, which I have found to be enormously flexible and easy to use. QUAD does not take full advantage of its flexibility, and I am looking for an application that does.

The Automated Office — Example: Producing a Newsletter

Eric A. Newcomer
Documentation Specialist
Criminal Justice Information Systems Division
Illinois Law Enforcement Commission
Chicago, Illinois

INTRODUCTION

The day of manual typing and filing of original letters, memos, and other short documents is coming to a close. The day of automated typing and filing will take its place.

Everyone knows that. The so-called "office of the future" has been the subject of countless articles, seminars, and sales presentations. Manufacturers such as Xerox, Savin, and Wang are busy producing and selling what they call "executive work stations."

These trends reflect the desires of managerial, professional, and executive personnel to join the data processing revolution. Most likely, this is the next step in the evolution of the office of the future.

This paper examines these trends in light of our experience with creating an "automated office" — providing computer capabilities to our professional, managerial, and executive staff. In brief, we found we:

- Reduced or eliminated paperwork and filing
- Used our resources more efficiently
- Saved on personnel costs
- Increased productivity.

The how and why of these findings will be presented in the following pages. An example of the way the automated office works is provided through a discussion of our method for producing a newsletter with the assistance of our computer. A discussion also is included of how this method is applied to produce some of our user documentation.

This paper is organized into the following seven sections:

- *Background Information.* This section provides background information on the Criminal Justice Information Systems Division.
- *Hardware Configuration.* This section briefly describes the hardware configuration in operation at the CJIS office.
- *File Group and Account Structures.* This section describes the file group and account structure in use on the CJIS HP3000.
- *Organization of the Automated Office.* This section describes the way the CJIS office became automated, the way it operates, and the way it should operate in the future.
- *Producing a Newsletter.* This section presents the example of the automated office in producing a newsletter with the assistance of the HP3000. Design tips are included.
- *Producing User Documentation.* This section describes how the method used to produce the newsletter can be used also to produce user documentation.
- *Conclusions and Observations.* This section presents some observations and conclusions about the automated office in general, based on our specific experience with it.

For the purposes of this paper, original text is defined as the document produced as a result of a person's desire to turn thoughts into written words.

I. BACKGROUND INFORMATION

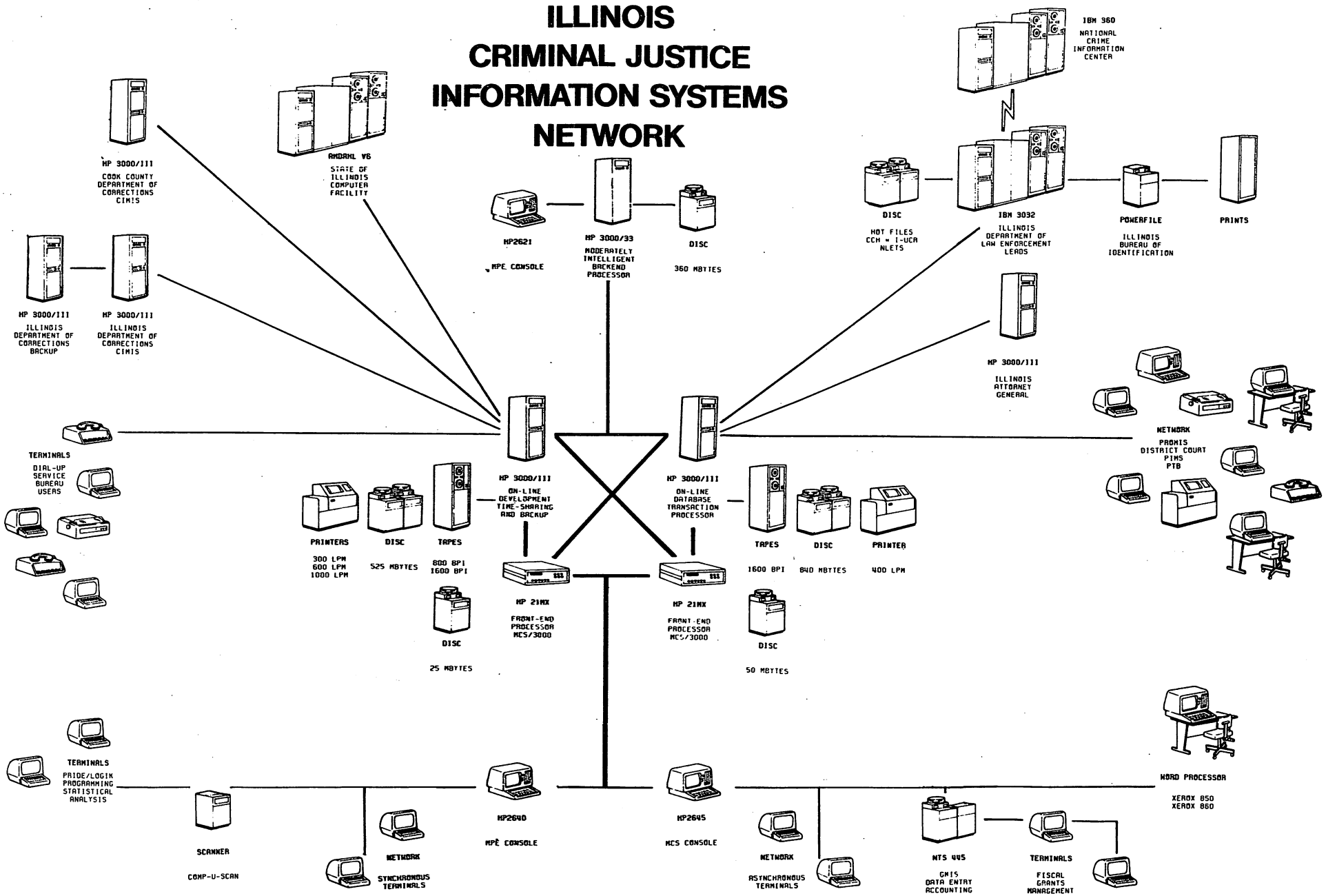
The Criminal Justice Information Systems Division (CJIS) of the Illinois Law Enforcement Commission functions as a computer consulting agency for other state and local Illinois criminal justice agencies.

CJIS also maintains a Statistical Analysis Center that develops statistical analysis methodologies and applies those methodologies to data collected by the software systems we design, as well as to data from other sources, such as the Uniform Crime Reports.

CJIS staff provide technical assistance to criminal justice agencies interested in acquiring data processing services and equipment. CJIS also designs, develops, and implements transaction-driven, real-time management information systems for state and local criminal justice agencies.

Recently CJIS developed and implemented an electronic transfer of inmate data between the Cook County Department of Corrections and the Illinois Department of Corrections. Both agencies use CJIS's Correctional Institution Management Information System (CIMIS) to collect and maintain their inmate data. The electronic transfer is timed to coincide with the weekly transfer of

Figure 1. CJIS hardware configuration



inmates from the Cook County Jail to the Joliet Correctional Center. The inmates' records arrive before they do.

Other systems currently under development include one for the Cook County State's Attorney's Office, one for the Police Training Board, one for the Illinois Attorney General, one for small and medium-sized Illinois police departments, and one to monitor the activities of juvenile detention centers throughout the state.

CJIS's 45 employees include 20 technical people, nine professionals, eight managers, and eight clerical staff.

II. HARDWARE CONFIGURATION

CJIS hardware configuration is represented by Figure 1. We operate two HP3000 Series IIIs, one for system development and back-up, and the other for system production.

Our in-house computer functions are handled by the development computer. Our on-line users are handled by the production computer.

Our office hardware configuration is centered around the development computer. We use about 35 CRT terminals for input operations, software testing, and system demonstrations. About 30 operate under MPE. The remainder are block-mode terminals used to test or demonstrate the systems we develop.

Of the 30 MPE terminals, about half are used by programmers and analysts; the other half by managers and professionals.

We have an optical-character-recognition scanner, but find it more efficient to type original text directly onto the computer using a CRT terminal.

For output we have two HP upper-case drum line printers, an upper and lower case dot-matrix Printronix line printer, two Agile 4210 letter-quality daisywheel printer/terminals, two Xerox daisywheel word processors with local storage and reformatting capabilities, and a Versatec 1200A electrostatic printer/plotter. Figure 2 illustrates this configuration.

Documents input on any one of the 30 or so CRTs can be output on any one of these output devices.

A person typing a letter or other document has the option to direct it to a line printer, to tell a word processor operator to print it out, or to use one of the AGILE printer-terminals to print it out himself.

As you can see from Figure 1, our HPs are connected to all sorts of other computers. This brings up an interesting sidelight to the discussion on the automated office — the transmission and reception of text and documents across communication lines. Already certain of our staff send and receive messages to and from Springfield, Washington, D.C., the Cook County Jail and the Illinois Department of Corrections.

Soon managers and professionals will enjoy the benefits of this kind of communication technology. Today's electronic mail and electronic database systems are only a small indication of the sort of assistance to come.

III. GROUP AND ACCOUNT STRUCTURE

Our group and account structures make allowances for text and document processing.

Most of the accounts on the HP are set up according to the software systems under development. The programmers and analysts working on the Police Information Management System (PIMS), for example, log on the PIMS account. The Attorney General's system people log on the AG account. And so on.

Each of these accounts includes a group specifically set aside for documentation and text files, usually called "DOCUMENT."

Managers and professionals unconnected with any one particular software system log on a generic "CJIS" account. Within this account are two groups set aside for text files — one for software documentation, and one for short documents such as letters, memos, and reports. The word processor operators log on this group, the "DOCUMENT.CJIS" group.

```
**********************************************************************
*                                                                    *
*                                                                    *
*                                                                    *
*                          --EXAMPLE 1--                             *
*                                                                    *
*                                                                    *
*    Word processor operators sign on:                               *
*                                                                    *
*      HELLO <name>/<password>.CJIS                                  *
*                                                                    *
*    and are automatically placed in the DOCUMENT group.             *
*    Managers and professionals are provided with similar            *
*    log on procedures.                                              *
*                                                                    *
*                                                                    *
**********************************************************************
```

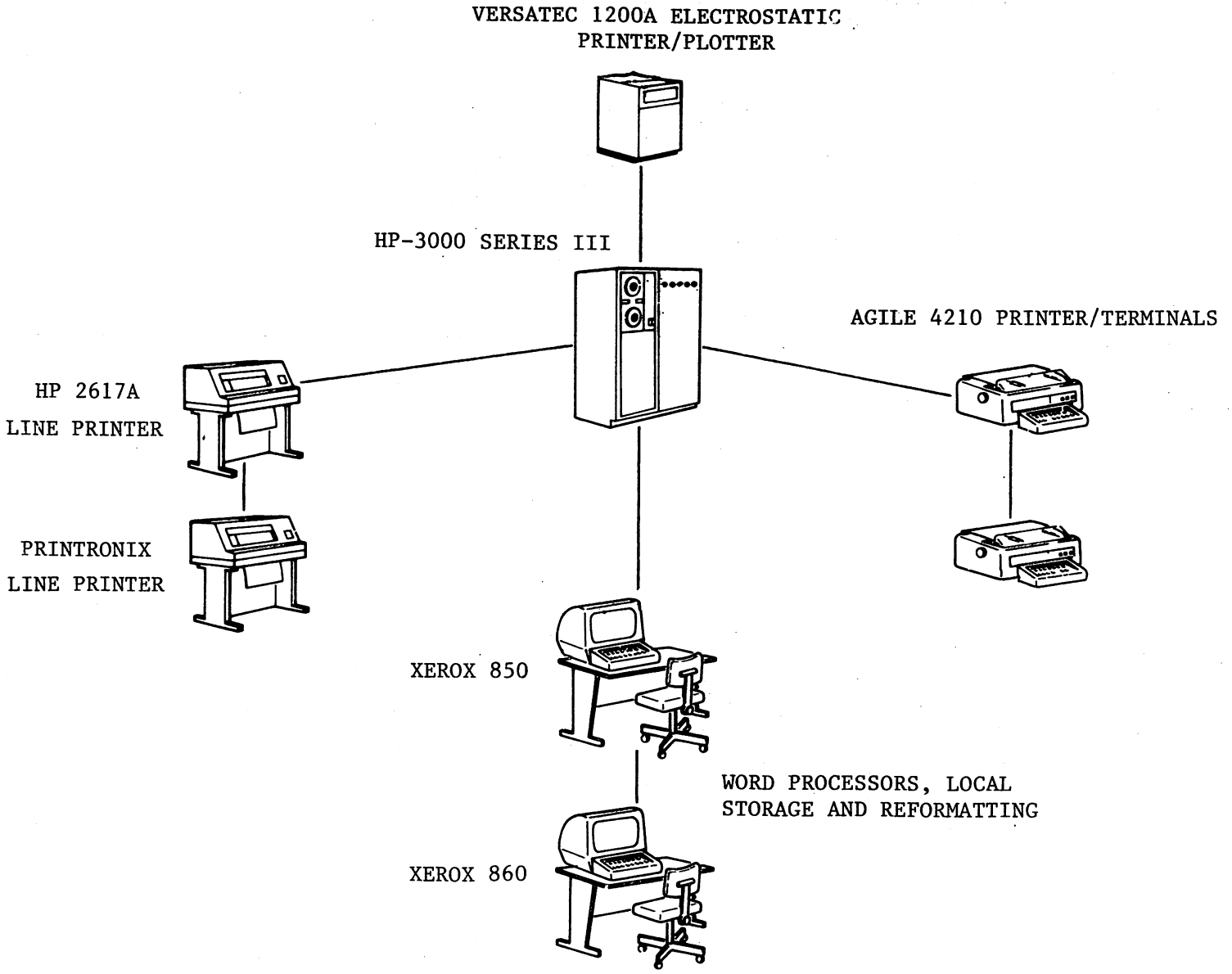


Figure 2. A person creating a letter, memo or other document has the option to direct it to a line printer, to tell a word processor operator to print it out, or to use the Agile printer/terminal to print it out himself. The plotter is used to produce charts and graphs.

This structure simplifies filing and recall of documents by segregating text files into separate groups within the various accounts. Furthermore, it provides a separate group within the "administrative" account specifically for short documents such as letters, memos, and brief reports.

This provides ease of permanent storage on magnetic tape. The whole group of files can be stored at once each month to provide a permanent record, if desired. We generally rely on twice-weekly system dumps and daily date dumps to provide back-up.

So far, this account structure is the only formal method of organizing and maintaining text files we use. Each office user is responsible for preserving, naming, and maintaining his or her own files within these groups, and is responsible for ordering separate magnetic tape storage, if any. Generally speaking, the system storage and dump procedures are reliable enough so that these back-up and permanent filing procedures aren't used very often.

In the future some sort of additional text file structuring according to function may be implemented to keep on file letters, memos, and reports of interest to future employees, or that provide historical reference.

As of today, the account structure allows the personnel to use the text files to supplant and supplement their own personal files, and to facilitate and direct document flow between personnel inside and outside the office.

IV. ORGANIZATION OF THE AUTOMATED OFFICE

During the past three years, our programming and analyst staff tripled. Our statistical staff doubled. Our clerical staff did not increase.

Yet we are getting more done, more quickly, and more efficiently. We are using the computer to eliminate wasteful and redundant paperwork and filing procedures.

Part of the reason CJIS professionals and managers use the computer is because our boss, CJIS Director J. David Coldren, uses one in his office. We were provided with a top-level example of how it could help.

When office workers find out how the using the computer can help them with their jobs, they ask to have terminals installed in their offices. They notice for themselves how using the computer can help them get their work down faster and more efficiently.

This fact was brought home to us by the realization that we never seem to have enough terminals to go around, no matter how many we order. Someone else is always asking for one.

Now that we've recognized the trend, we're in the process of studying and evaluating it, and planning for the future.

The way it works is illustrated by the following:

An administative assistant finds he must handle and generate a great deal of paperwork to fullfill the functions of his job. Letters, memos, brief reports, budget statements, etc.

He has a CRT terminal on his desk, which he requested about a year ago when he realized how much the computer could help him with his work.

One of his tasks is to prepare and distribute monthly progress reports on all CJIS activities. He created a text file on the computer using the TDP/3000 text processing subsystem. He keeps this text on file, calling it up each month to change only those parts of the report that require updating. He has entered formatting commands once, and shouldn't have to enter them again.

Each month he enters the changes, stores the new file, and tells the word processor operator the name of the new file. She prints it out, makes copies, and distributes it.

He rarely asks for draft copies anymore, so confident is he of his abilities to correctly type in, proofread, and format the monthly report.

```
--EXAMPLE 2--

To create a text file for a memo the manager types:

   :RUN TDP.PUB.SYS

   /ADD

   1     text . . .
   2     . . .
   3     . . .
   4     etc. . . .
```

```
*    /KEEP MEMO,UNN                                                *
*    /PRINT ALL, OFFLINE   (to get a working copy)                 *
*    /MODIFY 1/4  . . .                                            *
*    /KEEP                                                         *
*    /EXIT                                                         *
*                                                                  *
*    :RELEASE MEMO                                                 *
*                                                                  *
*                                                                  *
*  To modify a text file for a memo the manager types:             *
*                                                                  *
*                                                                  *
*    :RUN TDP.PUB.SYS                                              *
*                                                                  *
*    /TEXT MEMO                                                    *
*    /MODIFY 1/4  . . .  (etc.)                                    *
*    /KEEP                                                         *
*    /EXIT                                                         *
*                                                                  *
*    :RELEASE MEMO                                                 *
*                                                                  *
*                                                                  *
*  The manager tells the word processor operator to print          *
*  a file called "MEMO" on the standard memo form.  She            *
*  logs on and types:                                              *
*                                                                  *
*                                                                  *
*    :RUN TDP.PUB.SYS                                              *
*                                                                  *
*    /LISTQ MEMO                                                   *
*    /EXIT                                                         *
*                                                                  *
*    :BYE                                                          *
*                                                                  *
*                                                                  *
*  She stores the file on a floppy disk, and prints it out.        *
*  TDP formatting commands would not be used in this case.         *
*  If any reformatting were necessary, the word processor          *
*  operator would do it on the word processor.                     *
*                                                                  *
*                                                                  *
********************************************************************
```

When this administrative assistant replies to a letter of inquiry, he types his response into a text file, edits it, and stores it. He tells the word processor operator the name of the file, and instructs her to print it out on letterhead. She does so, addresses the envelope, and brings the letter to him for his signature.

Again, rarely does he request a draft copy to check, except for the most delicate and important letters.

What's eliminated in terms of paperwork is the draft; what's eliminated in terms of filing is filing working copies of the monthly report.

```
********************************************************************
*                                                                  *
*                                                                  *
*                                                                  *
*                          --EXAMPLE 3--                           *
*                                                                  *
*                                                                  *
*  The manager also can go to one of the Agile printer/            *
*  terminals and print out the memo himself.  He types:            *
*                                                                  *
*                                                                  *
```

```
*    :RUN TDP.PUB.SYS                                                *
*                                                                    *
*    /SET TERM AGILE15                                               *
*    /FINAL FROM MEMO                                                *
*    /EXIT                                                           *
*                                                                    *
*                                                                    *
*  To do this he would add the following TDP formatting             *
*  commands to the beginning of his text file:                       *
*                                                                    *
*                                                                    *
*    \LFT 10                                                         *
*    \RHT 75                                                         *
*    \TOP 12                                                         *
*    \BOTTOM 12                                                      *
*                                                                    *
*                                                                    *
*  This prints standard 65-character margins in 10 pitch,            *
*  leaving 1" on each side of the paper, and 2" margins at the       *
*  top and bottom of the page.                                       *
*                                                                    *
*  TDP provides automatic paragraph compaction and optional          *
*  hyphenation.  Options also are available to print in 12           *
*  pitch or proportional space (using the Agile or the Xerox),       *
*  force page feeds (\NEW), double-space, indent, underline,         *
*  and specify headings and page numbers.                            *
*                                                                    *
*  And he has a final-quality copy of his text file in hand.         *
*  He gives this to a secretary to photocopy and distribute.         *
*                                                                    *
*                                                                    *
**********************************************************************
```

Another illustration comes from the Statistical Analysis Center. An analyst recently completed two lengthy reports based on information gathered from the Cook County CIMIS we designed.

He typed the reports onto computer text files, complete with tables. He used the interactive and batch statistical analysis programs to formulate his results, and used our interactive graphics program to produce graphs and charts to illustrate his findings.

He formatted the document himself using the EDIT2 subsystem. He printed out draft copies using EDIT2 on the Agile printer/ terminal. A word processor operator will print out the final copy, using the same text file and the same text-processing subsystem.

```
**********************************************************************
*                                                                    *
*                                                                    *
*                          --EXAMPLE 4--                             *
*                                                                    *
*                                                                    *
*  To format the long file, to print drafts on the Agile and         *
*  finals on the Xerox 860, the statistician preparing the           *
*  report used the following EDIT2 commands:                         *
*                                                                    *
*                                                                    *
*    >>SET PAGESIZE=66                                               *
*    >>SET TOPSPACE=2,6                                              *
*    >>SET BOTTOMSPACE=2,6                                           *
*    >>SET HEADING=OFF                                               *
*    >>SET FOOTING=CENTER,#PAGE                                      *
*                                                                    *
*                                                                    *
*  This sets the pagesize to 66 lines, or 11 inches, sets the        *
*  page numbers to print in the center, and leaves margins at         *
```

```
*  the top and bottom of each page of 1-1/2 inches (9 lines).   *
*  EDIT2 leaves a blank line for the heading when it's turned   *
*  "off," and uses one line to print the page number at the     *
*  bottom of the page.  These commands provide room for 48      *
*  lines of text on each page.                                  *
*                                                               *
*  Each time the statistician wishes to force output to the top *
*  of a new page, he enters:                                    *
*                                                               *
*                                                               *
*     .NEWPAGE                                                  *
*                                                               *
*                                                               *
*  The ".NEWPAGE" command is executed through the PRINT         *
*  command when output is produced.                             *
*                                                               *
*  This will set up the file to print on the AGILE.  EDIT2 is   *
*  different than TDP in that some of the formatting commands   *
*  are entered as workfile options, instead of entered into     *
*  the text file.  Some defaults, such as page size (60) also   *
*  are different.                                               *
*                                                               *
*  To accept this formatted output on the Xerox 860's local     *
*  storage floppy disks:                                        *
*                                                               *
*                                                               *
*     o  The margins are set to 10 and 75 (EDIT2 defaults)      *
*                                                               *
*     o  Pagesize and pagelength are set to 66 to match         *
*                                                               *
*     o  Pitch is set to 10 (computer default)                  *
*                                                               *
*                                                               *
*  The word processor operator logs on, accesses the EDIT2      *
*  file, and executes a PRINT command.  The report is printed   *
*  onto the floppy disk exactly the way the statistician wants  *
*  it -- exactly the way the word processor will print it out.  *
*                                                               *
*                                                               *
*****************************************************************
```

These procedures were implemented using the EDIT2 and TDP subsystems developed by HP. We also use the QEDIT subsystem developed by Robelle Consulting. In the near future we hope to convert all text processing to HP's new TDP/3000. We've found it to be a more powerful, versatile subsystem. It'll also save CPU time and file space hogged by EDIT2.

In the future we plan to make more use of secretaries to input original documents and to make changes to existing ones, to use the Agiles for final output, to implement the sheet-feeder function on one of the Agiles to produce documents on letterhead, and to develop and implement formatting programs and use files that will automatically format and print various types of documents.

CJIS Director J. David Coldren recently developed and implemented a new SPOOLER for use with output files. The new SPOOLER allows files to be printed automatically on specific device types, such as line or character printers, and to programatically control the input and output of text files.

The new SPOOLER will allow us to take text input from any user, format it according to document type (such as letter, memo, or report), and automatically print it out.

The documentation specialist will design, establish, and implement standards for formatting and organizing documents, and write subroutines and use files to automatically produce those documents. A person would then enter the necessary text, and issue the command to print it.

A combination of the capabilities of the TDP subsystem and the Agile printer/terminals will allow us to produce documents with a minimum of formatting work and a maximum of standardization.

No operator would be necessary, other than regular computer room operators.

This is what we're working toward, and this is what the new text processors are increasingly allowing us to do.

--EXAMPLE 5--

The Old Way	The New Way
1. Write out by hand	1. Type into text file
2. Get WP to type draft	2. Get WP to print final
3. Proof & correct draft	
4. Get WP to type corrections	
5. Proof revision	
6. Get WP to print final	

The New Way eliminates draft copies.

This saves the WP operator time, which she can spend printing other people's documents, and saves personnel costs because one WP operator can print out as many documents as two or three can type and print out. It takes the manager about the same amount of time to create an original document each way, and less for updates to periodically-issued documents which only need revision.

V. PRODUCING A NEWSLETTER

The computer is used to produce charts and graphs to illustrate articles from the Statistical Analysis Center, to gather, edit, and produce copy ready for the printer's camera, to produce review copies of the articles, and to keep the articles on file in case they're ever needed again. See Figure 3.

An interactive graphics package CJIS developed makes use of our Versatec electrostatic plotter to produce line, bar, and pie charts, shaded maps of the state of Illinois, and Hudson algorithm graphs. HP Versaplot and Fortran software are used.

Staff members are assigned to write articles about projects they work on when there's a new development such as a new release of software or a new report.

An editorial board was formed of the agency's supervisors and managers, who assign articles and approve final drafts for publication. Staff members are responsible for placing their articles in text files, and releasing those text files for access by the editor.

The editor edits them for style and grammar only, a much less time-consuming and complicated procedure than editing them for content as well, which he used to do before the procedure was computerized.

The authors' text files, residing in their home accounts and groups, are copied over into one big text file in the editor's home account and group for editing, formatting, and printing out.

The basic idea is the editor works on-line, as do the authors of the articles. The authors enter the articles they write into text files, print them out for review by their supervisors, and when approved, release them to the editor. The editor edits them on-line using TDP, and makes use of various output options to assist him.

He uses the line printer to produce drafts with line numbers for editing, the Agile to produce review drafts for approval, and the Xerox 860 to reformat the articles into 3"-wide justified columns of proportional space type for paste-up.

In addition, authors generate graphs and charts to illustrate their articles, or the editor generates them on-line using the Versatec.

--EXAMPLE 6--

To gather and output a copy of the articles requires the following TDP commands:

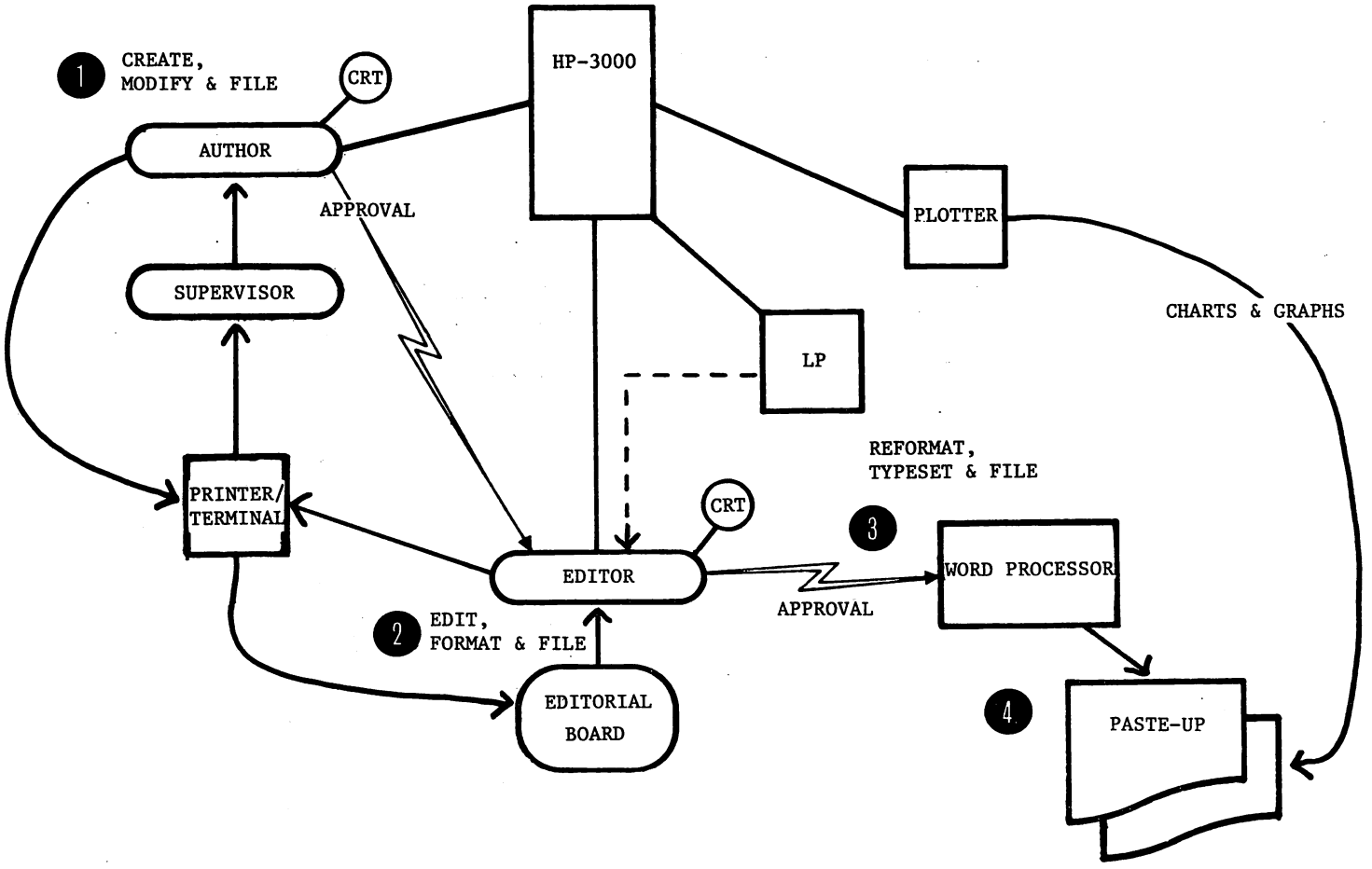


Figure 3. Steps in producing a newsletter with the assistance of the computer. (1) The author assigned to write an article creates a text file and prints out a copy for the supervisor's approval. Rewriters are done with the assistance of a text-processing subsystem. (2) The editor copies the article into a larger text file with other articles, prints out a working copy with line numbers on the line printer and prints out a copy of the edited file for review by the editorial board. (3) The approved articles are copied onto a local file on the word processor, reformatted, and typeset into justified columns. (4) Charts and graphs produced by the plotter are added during paste-up as illustrations for some of the articles. Variations of the procedure can be used to produce user documentation and other office documents and reports.

```
*     :RUN TDP.PUB.SYS                                                *
*                                                                     *
*     /TEXT ARTICLE1.GROUP.ACCOUNT                                    *
*     /JOIN ARTICLE2.GROUP.ACCOUNT                                    *
*     /JOIN ARTICLE3.GROUP.ACCOUNT                                    *
*     /JOIN . . . . . (etc.)                                          *
*                                                                     *
*                                                                     *
*  To produce a copy with line numbers for editing, modify the        *
*  file, and store it:                                                *
*                                                                     *
*                                                                     *
*     /LIST ALL, OFFLINE                                              *
*     /MODIFY . . . . (etc.)                                          *
*     /MOVE . . . . (etc.)                                            *
*     /REPLACE . . . . (etc.)                                         *
*     /KEEP EDITFILE,UNN                                              *
*                                                                     *
*     /EXIT                                                           *
*                                                                     *
*                                                                     *
*  The editor then goes to one of the Agile printer/terminals         *
*  and prints out copies for review.  He issues the following         *
*  format commands (these can be typed in on-line or stored           *
*  at the beginning of the file itself):                              *
*                                                                     *
*                                                                     *
*     \LFT 10                                                         *
*     \RHT 65                                                         *
*     \LINESPACE 2                                                    *
*     \PAGENO 1,CENTER                                                *
*     \TOP 12                                                         *
*     \BOTTOM 12                                                      *
*     \COPIES 3                                                       *
*                                                                     *
*                                                                     *
*  This will print out three copies of the edited articles,           *
*  with 55-character margins, double-spaced to provide plenty         *
*  of room for notes and corrections.  Two inches are left            *
*  at the top and bottom of each page.  The default pagelength        *
*  in TDP is 66.  TDP automatically fills or compacts the text        *
*  to fit within the specified margins.  The pagenumbers will         *
*  start with "1" at the bottom center of the page.                   *
*                                                                     *
*  Other commands allow the editor to do such things as affix         *
*  a heading displaying the time and date of the print-out, or        *
*  to indent or underline items of special importance.  These         *
*  commands can be found in the TDP manual, or produced on-line       *
*  through TDP's "HELP" command.                                      *
*                                                                     *
*                                                                     *
**********************************************************************
```

Articles reside in the home groups and accounts of the authors, in a file in the editor's home group and account, and on a floppy disk on the 860. If any question arises at any step in the procedure as to original wording, it can easily be resolved by printing out a copy of one of the files. If any file is accidentally lost or damaged, it can easily be replaced or recovered. The articles remain on file until purged by the authors, and until purged by the editor, generally about two months later. Any can be stored permanently on magnetic tape.

Furthermore, if the authors want to use their articles for other purposes, such as including them in a report or memo they're writing, they can use a text processing subsystem to copy their original files, modify them, or extract from them the sections they need.

If an article is rejected for one issue, it remains on file for the next one.

September, 1981

the Compiler

Volume 3, Number 2

Statistical Analysis Center ● Criminal Justice Information Systems ● Illinois Law Enforcement Commission
120 South Riverside Plaza, Chicago, Illinois 60606 (312) 454-1560

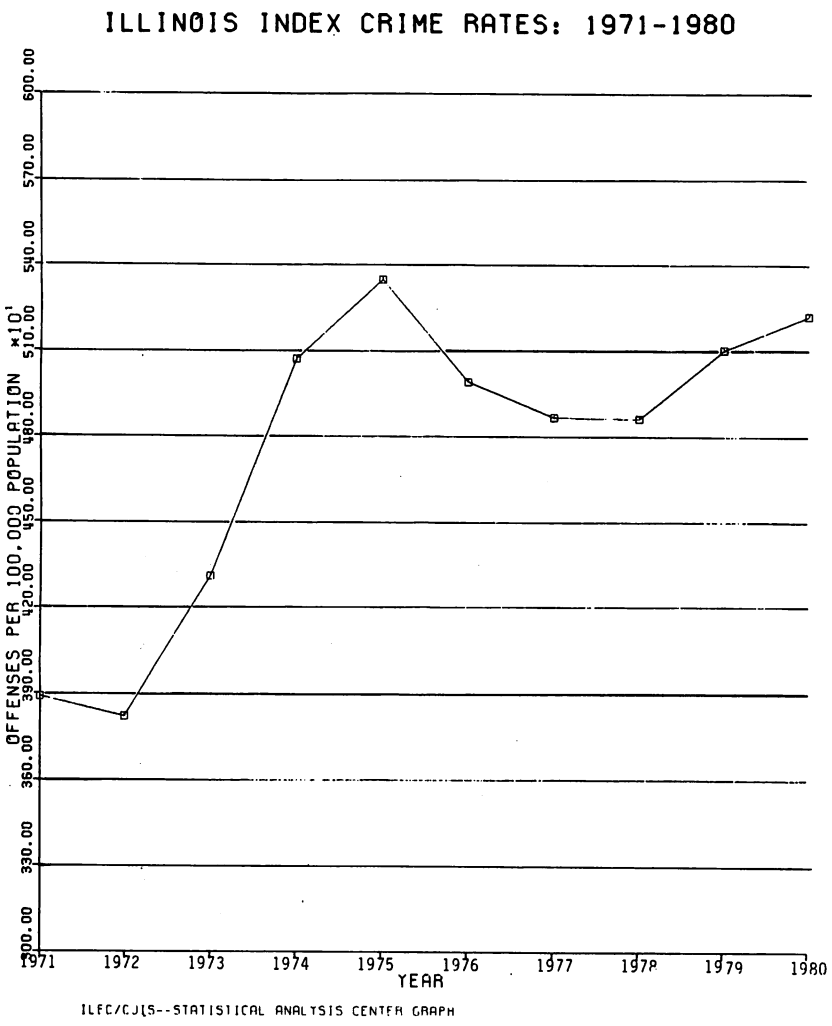


Figure 1. The graph shows the pattern Illinois index crime rates have followed from 1971 through 1980.

Correction

In the previous issue of the COMPILER (June, 1981) an article appeared on Page 3 entitled, "Council Audit Finds BOI Records Lacking." In that article we erroneously attributed to Bureau Chief Gary McAlvey, of the Department of Law Enforcement's Bureau of Identification, the statement that the County of Cook has not reported any felony dispositions to the Bureau since 1978.

A review of the transcripts of the Illinois Criminal Justice Information Council meeting at which we claimed McAlvey made that statement clearly shows that he did not. He indicated instead that the Bureau of Identification has not **posted** any felony dispositions received from Cook County since 1978. We extend our sincere apologies to both McAlvey and the Office of the Clerk of the Cook County Circuit Court for this error.

In the same article, we accurately attributed another statement to McAlvey that the County of Du Page is not currently reporting criminal case dispositions to the Bureau. However, on June 24, Clerk of the Du Page Circuit Court John Cockrell told the Council that there is a misunderstanding. The Bureau has been receiving dispositions from Du Page County but not entering them into the system due to technical difficulties.

We hope this correction and the story appearing on Page 3 sets the record straight.

Index crime in Illinois up 3.3 percent in 1980; up 9 percent in U.S.

by Larry Dykstra
SAC Analyst

Index crime in Illinois increased by 3.3 percent between 1979 and 1980. Violent crime increased by 4.2 percent, while property crime increased by 2.2 percent.

In comparison, index crime in America increased by 9 percent. Violent crime increased by 11 percent, and property crime increased by slightly more than 9 percent.

These figures are taken from recently-released 1980 Illinois Uniform Crime Report (IUCR) data and 1980 nationwide figures released by the FBI.

(continued on next page)

Figure 4. Sample cover of our newsletter shows type set using our word processor and graph produced using our plotter, both of which are connected to our HP3000 Series III.

If multiple copies of proof copies or lay-out copies are required, they can be produced simultaneously.

While the articles are being reviewed, the lay-out process can begin. Minor last-minute changes can be made on the word processor, and cut and pasted in. See Figure 4.

--EXAMPLE 7--

To reformat and typeset the articles on the Xerox 860 word processor, the WP operator logs on, and types in:

```
:RUN TDP.PUB.SYS

/LISTQ EDITFILE
/EXIT

:BYE
```

Now the text is on the floppy disk, unformatted except for the page breaks the word processor puts in automatically.

To reformat the text into 3"-wide, justified, proportional-space columns, the WP operator calls up the stored document, and changes its recordable format block options to the following:

Margins:	10 46	(36 chars. = 3" PS)
Pitch:	PS	(Proportional Space)
Justify:	X	(on)
KB/PW:	Standard	(from ASCII, used for comm.)

The operator exercises the reformatting software options available on the 860 to effect the changes. She marks the options she wishes to reformat:

- Margins
- Pitch
- Justify
- KB/PW (i.e., character set)
- Page Lay-Out
- Page Labels (i.e., because of narrow columns)

Entering these options initiates the interactive hyphenation routine. The WP operator hyphenates, stores the reformatted columns, and prints them out. The editor waxes the backs of them, cuts them out, and pastes them up.

The above options can be used in varying combinations to produce columns of virtually any size, type in different pitches, or ragged-right text.

In the future we'd like to increase the efficiency of the formatting commands, format the articles on the Agile, and eliminate the step of local storage and reformatting at the word processor (860). Eventually we'd like to format the articles for transmission to an on-line typesetting service, and figure out the entire lay-out in advance (how many pages, how long each article will be, what the sizes of the illustrations will be, etc.).

Articles are automatically filed for the authors, and may be produced at will. Word processors are used as output devices rather than input/output devices, and more efficient use is made of resources. Producing copies for review and proofreading and editing is a process of exercising format options of the text processing subsystems.

```
**********************************************************************
*                                                                    *
*                                                                    *
*                                                                    *
*                --NEWSLETTER PRODUCTION TIPS--                      *
*                                                                    *
*                                                                    *
*  Choose a two- or three-column design or a combination.            *
*  Figure out the width of your columns by figuring the size         *
*  of your page minus margins.  Work at 125 percent of original      *
*  size and have the lay-out sheets reduced to 80 percent at         *
*  the printer's.                                                    *
*                                                                    *
*  Size graphs, photographs, and drawings with a reduction           *
*  wheel.  Paste up on non-repro blue graph paper, using a           *
*  waxer to supply the paste and a light table to line               *
*  everything up.                                                    *
*                                                                    *
*  Use Letraset or Kroytype letters for headlines, Rapidographs      *
*  and Chartpak tape for lines, and cut and paste by-lines,          *
*  page numbers, and cutlines.                                       *
*                                                                    *
*                                                                    *
**********************************************************************
```

VI. PRODUCING USER DOCUMENTATION

Many of the same procedures and methods we use to produce the newsletter with the assistance of the computer we use also to produce some of our user documentation.

We use our letter-quality printers to prepare copy ready for the printer's camera. CRT screen forms are printed out by a special program that reads the various forms files and formats them for output on the Agile or the Xerox 860.

The programming, analyst, and managerial staff take part in the production of documentation by doing the initial writing for the systems they work on.

The documentation specialist texts the files from the others' accounts and groups into his own, edits them, does any additional writing that may be required (such as introductions and glossaries), and formats them for printing out.

He makes the changes in the documents, and has a secretary with a terminal enter them into a text file. He checks the file, then has the secretary or word processor operator print it out.

```
**********************************************************************
*                                                                    *
*                                                                    *
*                                                                    *
*                          --EXAMPLE 8--                             *
*                                                                    *
*                                                                    *
*  To format and print our user documentation, we use the            *
*  following TDP commands, embedded at the beginning of and          *
*  throughout the file where needed:                                 *
*                                                                    *
*                                                                    *
*    (:RUN TDP.PUB.SYS)                                              *
*                                                                    *
```

```
*    /FINAL FROM USERMAN                                                *
*                                                                       *
*    . . .                                                              *
*                                                                       *
*    \LFT 10                                                            *
*    \RHT 75                                                            *
*    \PAGENO "A-  1", CENTER                                            *
*    \PAGENOLINE 60                                                     *
*    \BOTTOM 10                                                         *
*    \TOP 12                                                            *
*    \HEAD "INTRODUCTION", RIGHT                                        *
*    \HEADLINE 6                                                        *
*    \IMAGE                                                             *
*                                                                       *
*                                                                       *
*  These commands result in 65-character output (6-1/2" wide),          *
*  with a top margin of 12 lines (2 inches) and a bottom margin         *
*  of 10 lines (1-2/3").  The heading will be printed flush             *
*  right, and reads "INTRODUCTION."  It will be printed on line         *
*  6, leaving an inch before the start of the text.  The page           *
*  numbers will start with A-1, and will appear at the bottom           *
*  center, 6 lines (1 inch) from the bottom.  The "\IMAGE"              *
*  command causes text to be printed out exactly the way it             *
*  was typed in.  The default setting on TDP, "\FORMAT",                *
*  compacts and compresses text.  Our documentation follows             *
*  a strict format, and the FORMAT setting can easily distort           *
*  it.                                                                  *
*                                                                       *
*  New pages are specified by "\NEW" commands imbedded in the           *
*  text.                                                                *
*                                                                       *
*  New page numbers are specified by a \PAGENO "B-  1" command,         *
*  for example.                                                         *
*                                                                       *
*  New headings are specified by a \HEAD "SECTION 1", RIGHT             *
*  command, for example.                                                *
*                                                                       *
*  TDP also provides a capability to automatically generate a           *
*  table of contents.                                                   *
*                                                                       *
*                                                                       *
************************************************************************
```

In the old days one person did everything, much like the newsletter. The new system saves research and writing time, and increases the amount of documentation than can be produced in a given amount of time.

In the future we'd like to extract more information from the computer, through our automated system design methodology. We'd also like to do more piecing together of new documentation from old documentation, making such things as glossaries and introductions standard.

VII. CONCLUSIONS AND OBSERVATIONS

The benefits of placing programmers on-line and allowing them to write code on-line are obvious and generally accepted. The benefits of placing managers, researchers, writers, and other office professionals on-line are just as obvious, but not as generally accepted.

The problem often comes up in this way: When confronted with the question of how office paperwork is handled, many managers respond, "That's what secretaries are for."

That's like saying, "That's what keypunch operators are for." It's realistic, but short-sighted.

The important thing is the product. That's what work is all about. Providing managers and professionals with on-line access to the computer results in more product in less time and at a smaller cost, just as providing programmers on-line access to the computer does.

Managers must produce the original documents of their office paperwork, just as programmers must produce the original code.

Hardware costs continue to decline while personnel costs steadily increase. The more use an office can make of its computer hardware and software to help generate its products, the less money that office will have to spend on personnel costs.

With a little training and practice, managers and professionals can create and enter original documents into text files in about as much time as takes for them to write them out by hand or dictate them.

For some this is a difficult process to get used to. There are always those who like their way of doing things and will not change. Even they can be made more productive by taking their handwritten or dictated documents and having a secretary enter them onto one of the HP's text editing subsytems.

The original document should be created on the computer, where it can be processed by text processing subsystems, directed to various devices for output, automatically filed, backed-up, and stored, and for all intents and purposes be considered permanent.

Once an original document is on the computer, what's done to it becomes a choice of electronically-controlled options. Those options include printing it out on the word processor.

--TEXT FILE OPTIONS--

- Creating Original Documents
- Automatic Filing
- Multiple Output Options
- Formatting and Editing Software
- Auditing and Back-up
- Large File Capacity
- Pass or Send File to Others.

Exercising these options on the computer eliminate draft copies and filing, save time and therefore personnel costs, and increase the amount of work that can be produced during a given amount of time. The document's on file if it's ever needed again, and can be automatically located. It can be used again, copied and modified, extracted from, or incorporated into a larger piece.

The word processor should be used to produce the final product as much as possible, not to input it.

That's what the word processor is for — nice output. Concentrate on getting as much nice output from it as possible.

The disadvantages of typing original text on word processors outweigh the advantages. Local storage devices such as floppy disks are unreliable; file structure is neither logical or centralized; and options for making use of the file in other ways are limited.

It's much better to connect the word processors up to the computer, type the original text on the computer, then print it out on the word processor.

--TO SUM UP--

- It takes the same amount of time (or less) for managers and professional staff to type short original documents into text files as it does for them to write them out by hand or dictate them.
- Word processors are better used as output devices than as input/output devices.

- Regular CRT terminals are good enough "executive work stations."
- It is more efficient to enter original text onto the computer and transmit it to a word processor than it is to type it on a word processor.
- Once an original document is on the computer, printing and filing it become matters of exercising options.

Perhaps in the future, we'll be able to place the letter-quality printers under the supervision of the regular computer operators, and print final documents semi-automatically, as line printers do.

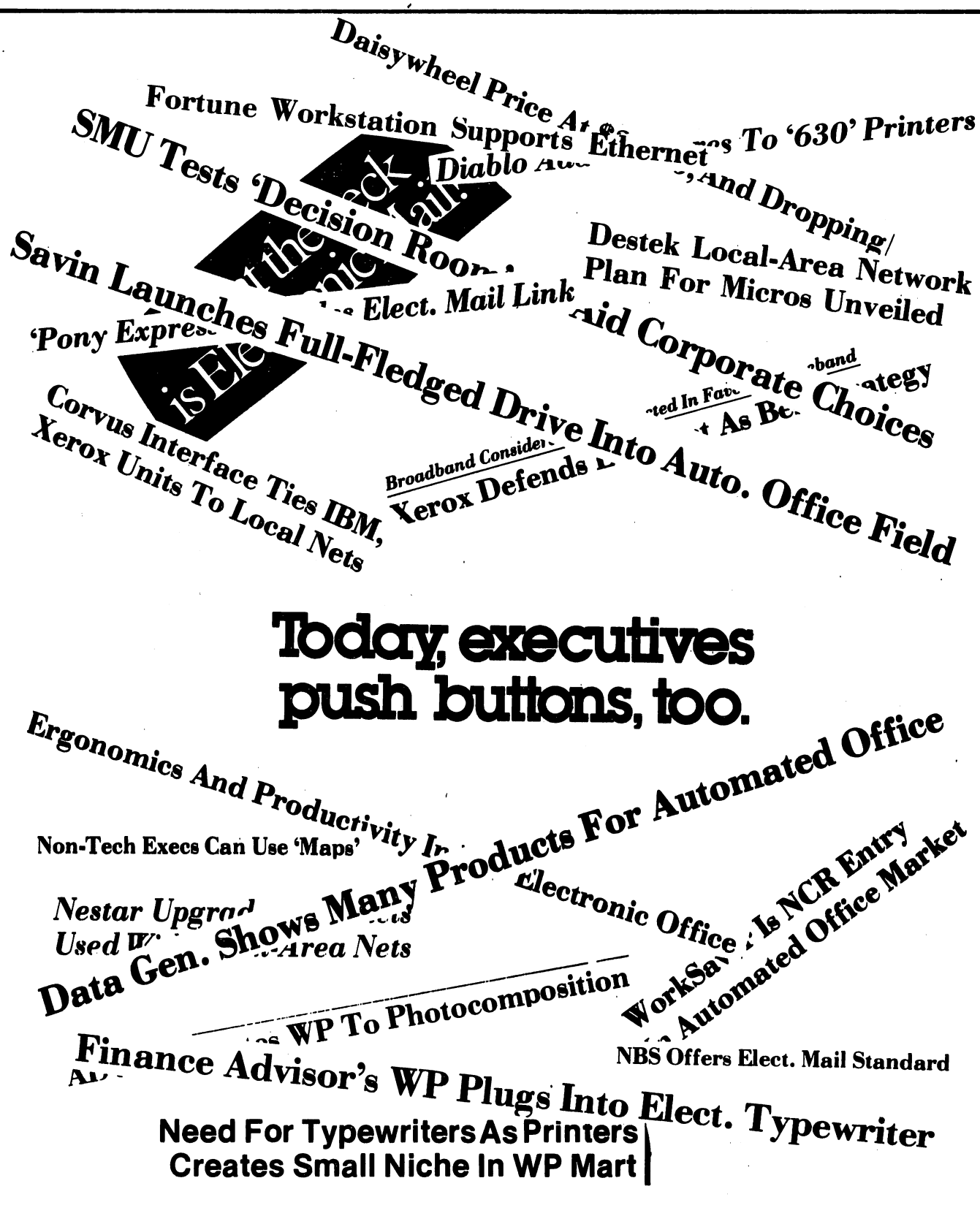
The office of the future will include a terminal on everyone's desk. The terminals will receive and display messages from around the world, and will accept input of messages, reports, communiques, tables, charts, and graphs for communication to somewhere else, local or otherwise. Much more paperwork and filing will be eliminated.

The small steps we have taken toward that eventuality bear this out. Our professional and managerial personnel clamor for convenient use of a terminal. Most now have them in their offices. Those who don't ask for them. They find they can do their work more efficiently on the terminal in the amount of time they used to spend getting someone else to do it for them.

The trend toward developing and marketing "executive work stations" is obvious. Wang came out with one. Xerox has one. Savin is coming out with one.

Xerox advertises theirs as "for business professionals, engineers, analysts, researchers," and so on. It handles electronic mail, aids in constructing tables, charts, and graphs, and handles filing, they say.

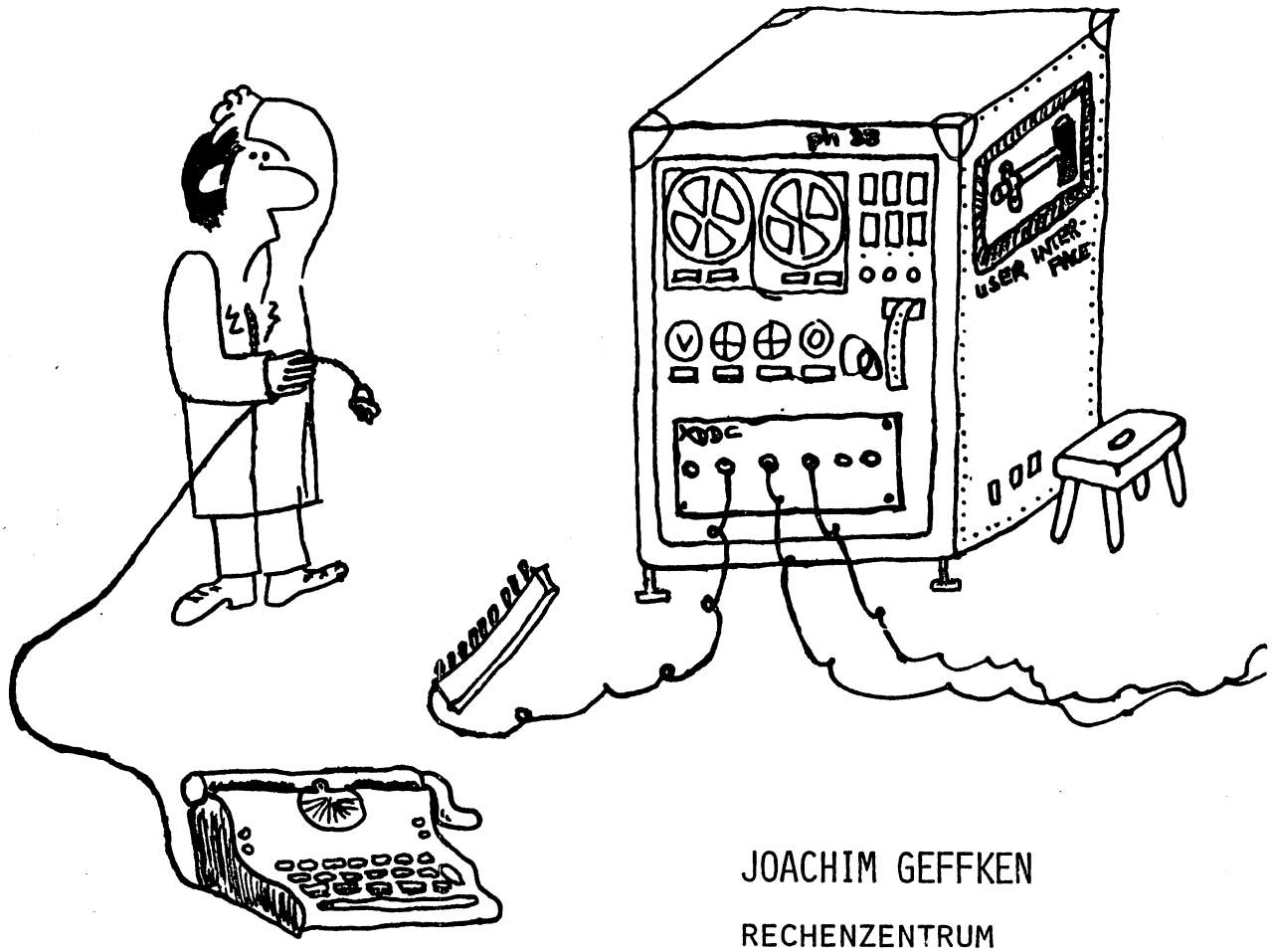
Well, that all sounds very impressive, especially to a new user. But the fact is we do all that with our HP3000, regular CRTs, and good software.



Headlines such as these reflect the growing trend toward providing computer power to managers and professionals to assist them with their office work.

Integrated Data- and

Textprocessing with hp3000



JOACHIM GEFFKEN

RECHENZENTRUM
HERBERT SEITZ KG
GRÜNENSTRASSE 11/12
2800 BREMEN 1

Survey

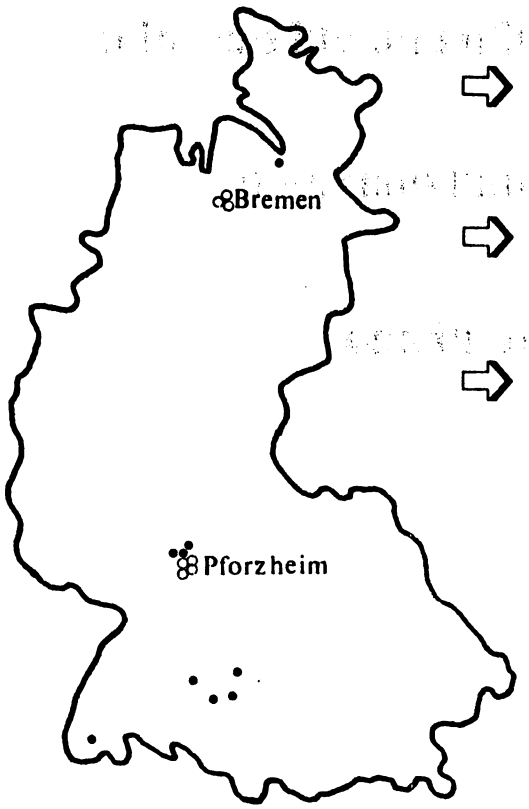
- ⇨ **Introduction**
- ⇨ **Fileaccess**
- ⇨ **Dataselection**
- ⇨ **Correction aid**
- ⇨ **Customizer**
- ⇨ **Supervisor**
- ⇨ **Interface**

Introduction

The Herbert Seitz Company is ...

- A REALTIME DATAPROCESSING SERVICE BUREAU
- AND SOFTWAREHOUSE
- AND HEWLETT PACKARD OEM

with (1981) ...



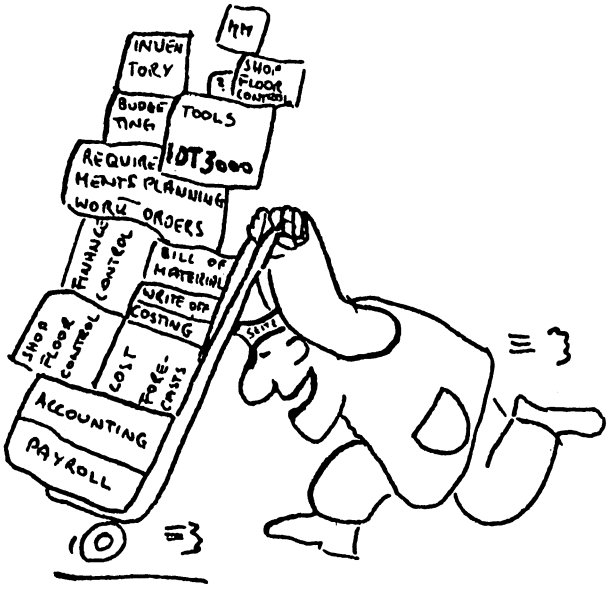
Location of:

- o own Computers
- • Associated Companies

- 7 OWN HP 3000 (SERIES III AND 44) IN OUR BREMEN AND PFORZHEIM BRANCH
- AND 10 HP 3000 SERIES III IN ASSOCIATED COMPANIES
- WITH APPROX. 350 TERMINALS SPREAD OVER GERMANY CONNECTED VIA HARD-WIRED LEASED LINES/DIALED LINES

Introduction 2

WE PROVIDE OUR SERVICES IN GERMANY AND FRANCE FOR COMMERCIAL APPLICATIONS LIKE ...



- ACCOUNTING
- PAYROLL
- MATERIAL MANAGEMENT
- SHOP FLOOR CONTROL, CAPACITY PLANNING
- TOOLS FOR HP 3000 OPERATION, SOFTWARE-DESIGN AND DOCUMENTATION

OUR USERS ARE ...

- WORKMEN
- DATA TYPISTS, CLERKS
- MANAGERS

ONLY A FEW OF THEM ...

- ARE SPEAKING (HP-)ENGLISH
- HAVE DP EXPERIENCE
- HAVE SEEN ANY TERMINAL BEFORE

Introduction 3

THIS PRESENTATION IS A GENERAL DESCRIPTION OF SOME TECHNIQUES OF INTEGRATED DATA- AND TEXTPROCESSING ON HP3000 COMPUTERS AS THEY ARE IMPLEMENTED IN IDT3000. THIS IS NOT A COMPLETE PRODUCT OVERVIEW.

IDT3000 IS A DATA- AND TEXTPROCESSING SOFTWARE PACKAGE DESIGNED BY HERBERT SEITZ KG WITH:

- AN IMAGE TEXTDATABASE AND DICTIONARY
- POWERFUL TEXTEDITING AND FORMATTING FEATURES FOR BUSINESS LETTERS AND REPORTS
- FILE ACCESS TO IMAGE-, KSAM- AND MPE-FILES
- A SELF LEARNING DICTIONARY AND AN ON-LINE CORRECTION AID
- MULTILINGUAL SCREENS, MESSAGES AND HYPHENATION ALGORITHMS
- A CUSTOMIZER FOR CHARACTER SETS, TERMINAL- AND PRINTERTYPES, DATAFILES, DATADEFINITIONS AND OPERATING ENVIRONMENTS
- INTERFACES TO DATAPROCESSING AND DATACOMMUNICATION
- A DAILY REPORT OF THE OUTGOING LETTERS AND REPORTS
- A TRACKING MECHANISM FOR RENEWED SUBMISSIONS
- A BATCH PROCESSING INTERFACE

CORRECTION AID / DICTIONARY (1)

GENERAL CONSIDERATIONS:

- THE USE OF A DICTIONARY MAKES ONLY SENSE IF THE VOCABULARY IS SUFFICIENT

- A VOCABULARY OF APPROXIMATELY 150.000 WORDS IS A REASONABLE COMPROMISE (VOCABULARY, DISC SPACE AND ACCESS TIME)

- A STATISTICAL EVALUATION OF THE VOCABULARY DURING THE SELECTION-PROCESS IS ADVISABLE

- THE GENERAL RULE OF THUMB FOR DATABASE CAPACITIES SHOULD BE KEPT IN MIND IN ORDER TO GAIN REASONABLE RESPONSE TIME (I.E. CHECKING OF 100 WORDS IN 2 - 3 SECS.)

- UNTIL THERE ARE BETTER ALGORITHMS AVAILABLE FOR PARSING, INTERPRETING AND UNDERSTANDING TEXT IN HIS CONTEXT IT IS NECESSARY TO MAKE THE DICTIONARY

⇒ USER ACCESSIBLE

AND

⇒ SELF LEARNING

SEE NEXT PAGES ...

CORRECTION AID / DICTIONARY (2)

Form 4 Maintenanance correction aid

word : Ⓐ

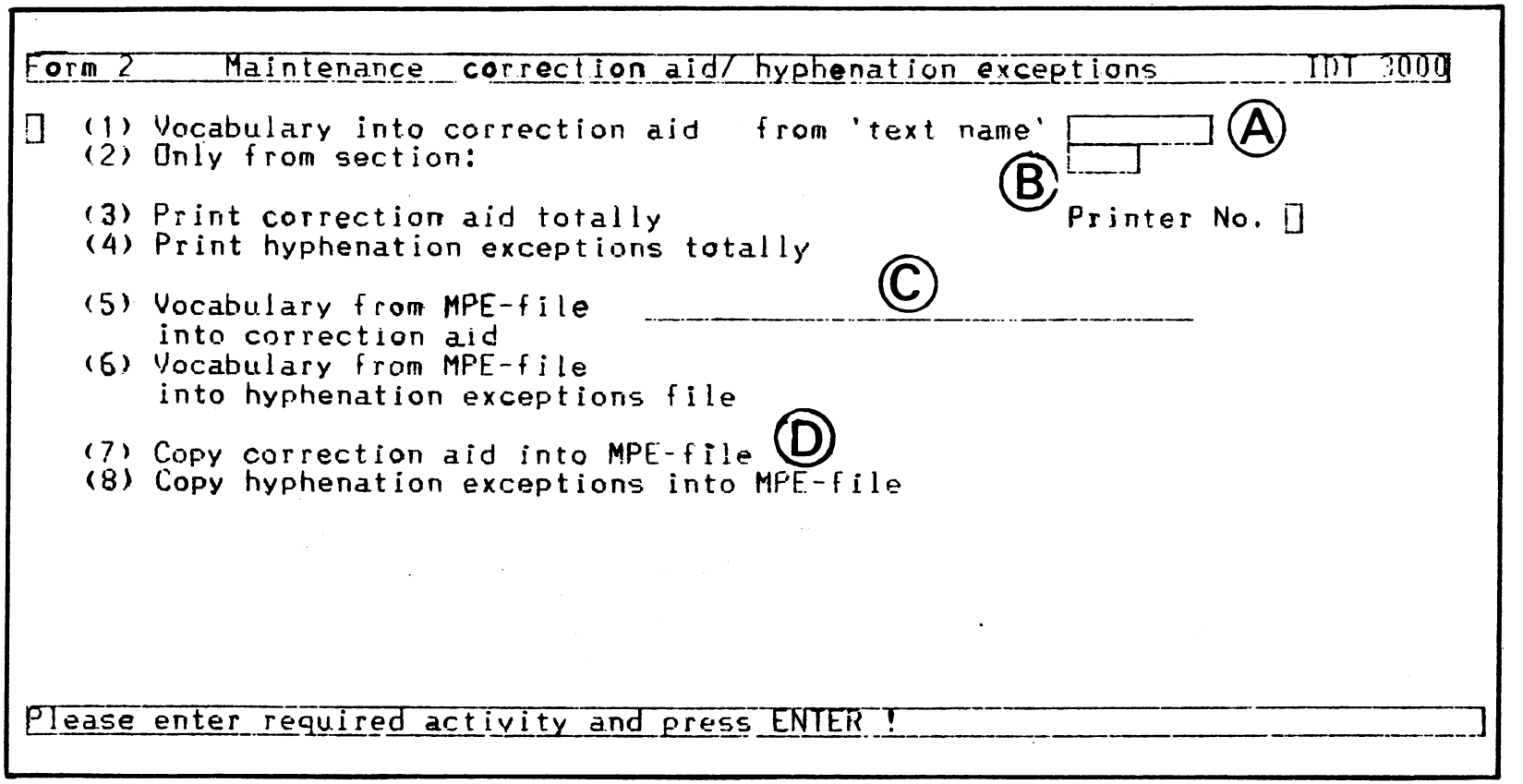
userfriendliness

☐ (X) Copy display to printer

Please enter new word for Your dictionary !

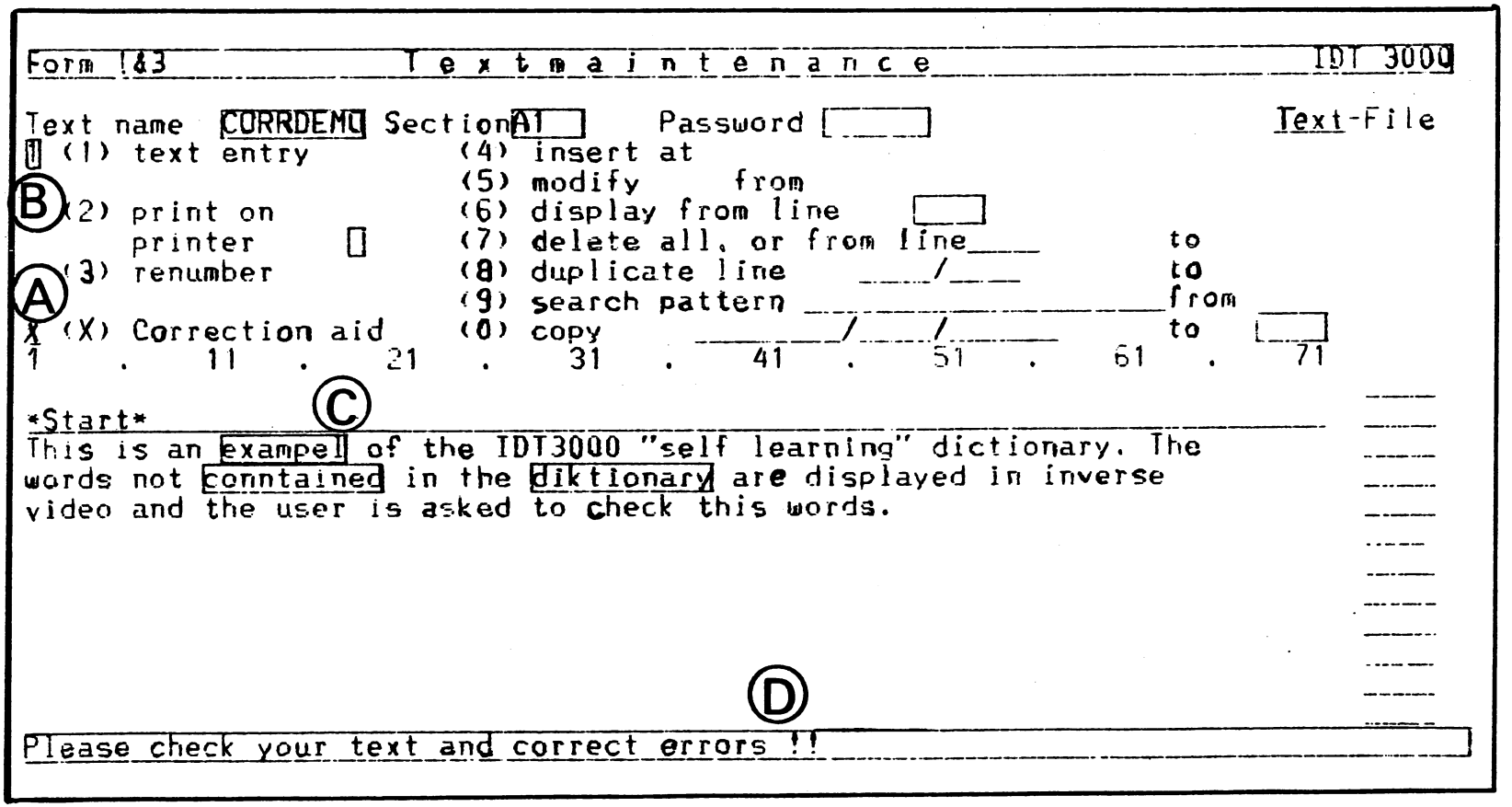
THE USER MAY ENTER OR UPDATE DICTIONARY ENTRIES WHENEVER NECESSARY Ⓐ.

CORRECTION AID / DICTIONARY (3)



NEW VOCABULARY MAY BE ENTERED FROM THE TEXTDATABASE (A) OR ONLY SOME SECTIONS (B) OR MPE-FILES (C). IF THE TEXT IS CAREFULLY CHECKED IN THE FIRST PERIOD OF USE THE DICTIONARY WILL BECOME MORE AND MORE SUFFICIENT FOR THE SPECIFIC APPLICATION. THE DICTIONARY MAY BE RESTORED TO MPE FILES (D) FOR BACKUP PURPOSES OR STATISTICAL EVALUATIONS.

CORRECTION AID / DICTIONARY (4)



THE CORRECTION AID CAN BE ENABLED THROUGH THE USER (A) AND WORKS DURING TEXTENTRY OR UPDATE (B). UNKNOWN (NOT NECESSARY WRONG) WORDS ARE MARKED (C) AND THE USER IS PROMPTED FOR RECHECKING AND CORRECTING (D).

CORRECTION AID / DICTIONARY (5)

```
Form 3          Maintenance of hyphenation-exceptions                IDT 3000

                                                                        (A)
Word                : ABBREVIATION
Hyphenation at      :    ^  ^

                    (^) Normal hyphenation after this column
                    (K) for  ck to k-k hyphenation  (special feature for german)
                    (V) for consonant duplication   (special feature for german)

existing definitions will be displayed


[ ] (X) Copy screen to lineprinter


Please mark where You want a hyphenation
```

THE USER MAY ENTER OR ALTER EXCEPTIONS FOR THE HYPHENATION ALGORITHM (A) WHEN NECESSARY. IN THIS WAY THE STANDARD PRECISION OF APPROX. 95% MAY BE INCREASED > 99% FOR A SPECIFIC APPLICATION WITH ITS TYPICAL SET OF VOCABULARY AND SIZES OF THE FORMATTED TEXT.

CORRECTION AID / DICTIONARY (6)

Form 6 Pattern Maintenance IDT 3000

No. Pattern (A) (B)

300 $(IDT 3000 the integrated Data- and Textprocessing Software$)

Existing patterns will be displayed !

☐ (X) Copy screen to lineprinter

Please enter or modify pattern!

COMPLEX TEXT EXPRESSIONS MAY BE DEFINED AS TEXT PATTERNS (A), IF NECESSARY WITH TEXT FORMATTING COMMANDS (B). PATTERNS ARE CALLED WITH THE COMMAND &nnn (nnn = PATTERN NO.).

FILEACCESS (1)

THE INTEGRATION OF DATA- AND TEXTPROCESSING REQUIRES:

- COMFORTABLE ACCESS TO DIFFERENT FILE TYPES (IMAGE, KSAM, MPE)

- USER ACCESS TO ALL FILES WITH READ ACCESS

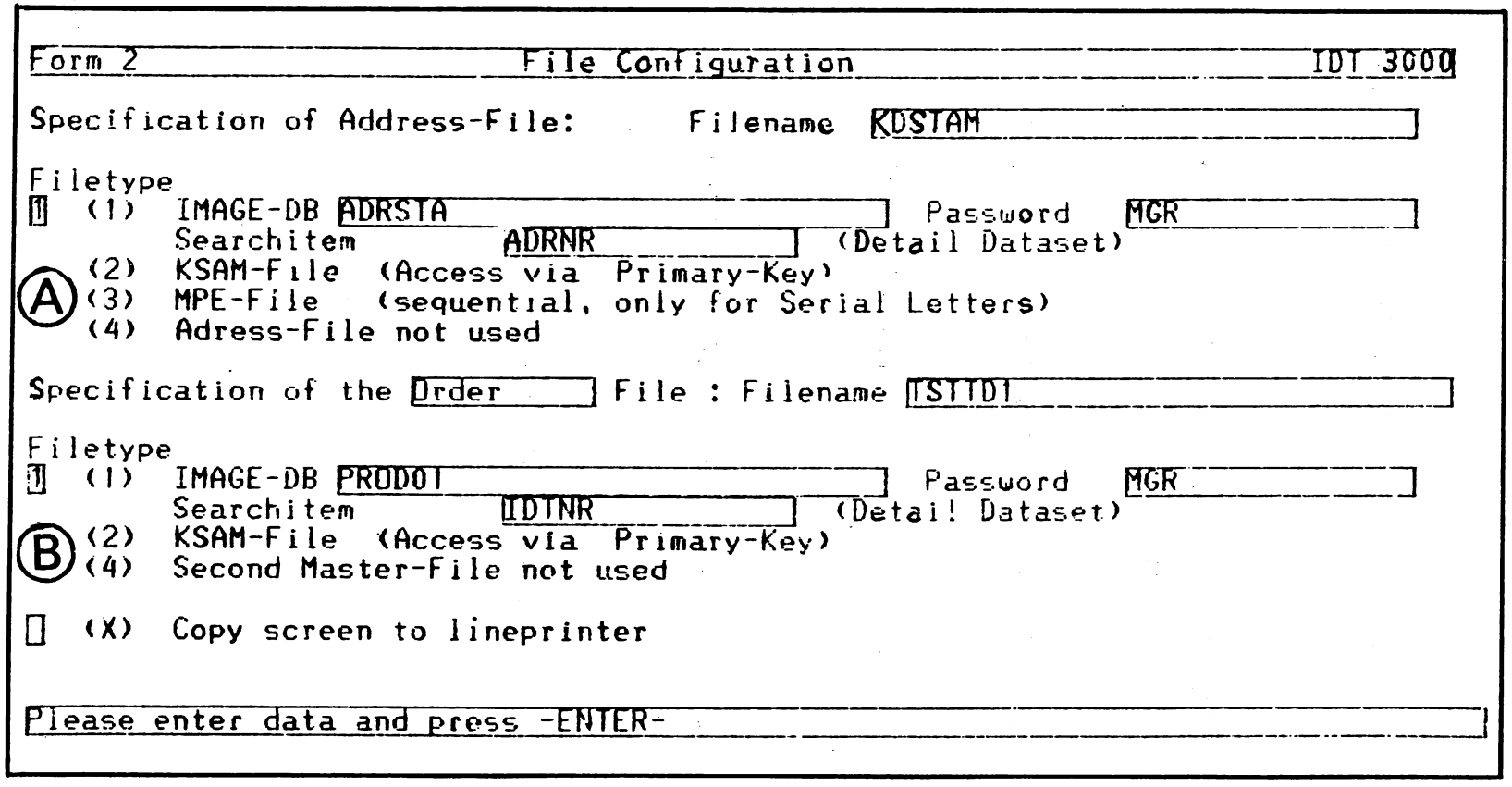
- UNLIMITED USER ACCESS TO THE CONFIGURATION OF THE OPERATING ENVIRONMENT

- EASY MODIFICATION OF DATA FORMATS AFTER LAYOUTCHANGES

- DATA ACCESS VIA DATA DICTIONARIES

SEE EXAMPLES NEXT PAGES ...

FILEACCESS (2)



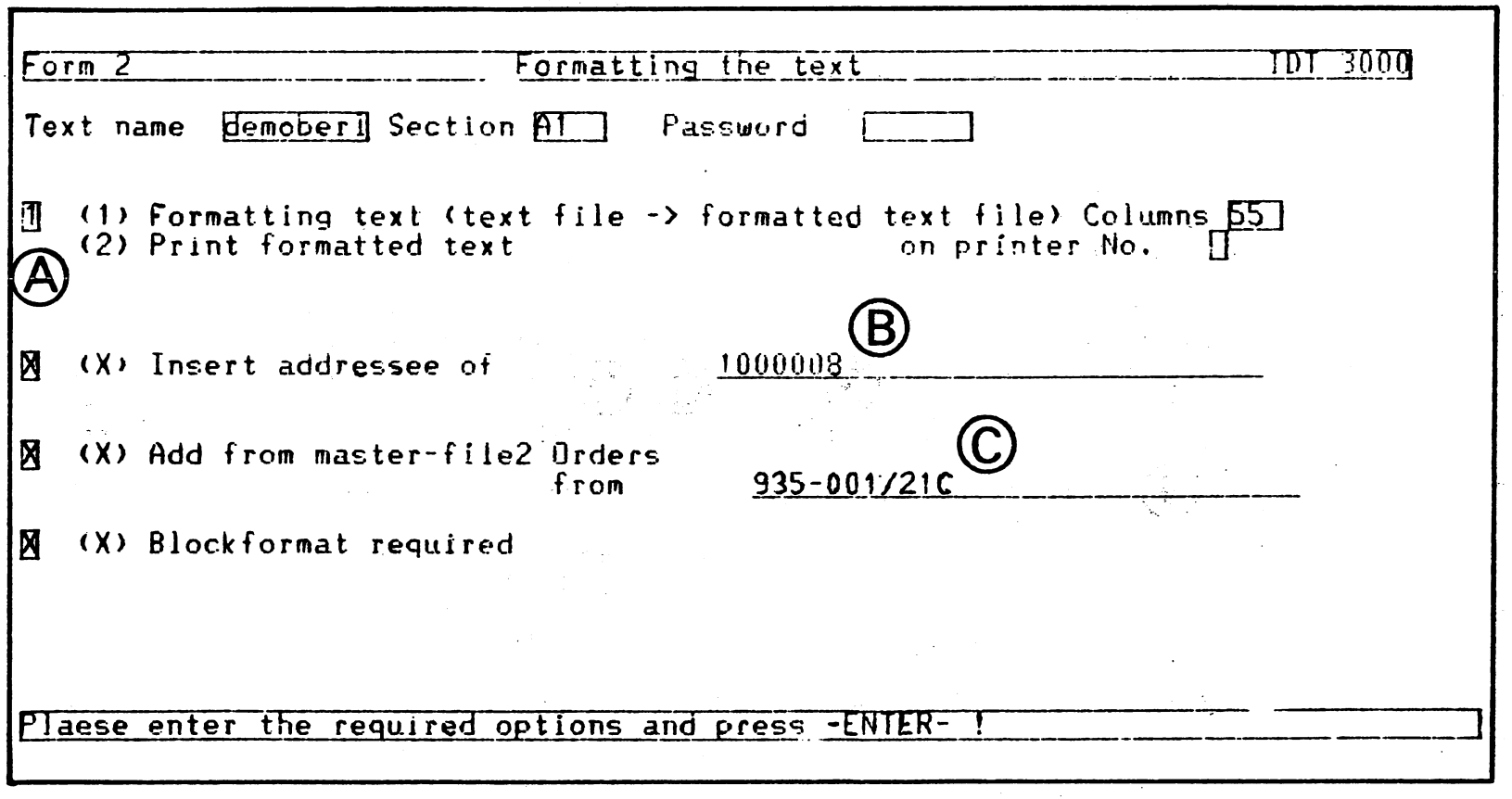
THE USER MAY DEFINE OR CHANGE FILENAME AND TYPE FOR AN ADDRESSFILE (A) AND ANOTHER USER SELECTABLE FILE (B).

FILEACCESS (3)

```
Form 5      Customizing data access (adress-file and master-file 2)    IDT 3000

A (A) Customizing adress-file          (B) Customizing master-file 2

In this form You may customize Your      Adress- file

You will have to describe each field (1 - 50).
Existing descriptions will be shown.

Field Fieldname (B)      Start Data-(1) Size(2) Format(3)  (F)
No.                      Col.  Type
12 (A) Name              015   [ ]      20     [          ]

(1) P4-P12 packed numeric item (C) (D)      (E) (3) 'X' place for 1 alphanumeric
    C2-C10 binary coded numeric item                character
    F      alpha-numeric item                   'Z' supress zeros

(2) Size :                                          the last digit must be
    you may need one digit for a                    an 'X' for a sign
    sign character!                                 character (numeric-item
[ ] (X) Copy screen to lineprinter
Please enter fieldformat!
```

UP TO 100 FIELDS PER USER AND/OR SESSION CAN BE CONFIGURED (A). THE NAME (B), POSITION (C), DATA TYPE (D), SIZE (E) AND OPTIONAL EDIT MASKS (F) CAN BE DEFINED AND MODIFIED WHEN NECESSARY.

FILEACCESS (4)



THE FILEACCESS (AND THE DATA SELECTION) CAN BE DONE DURING THE TEXT FORMATTING (A). DATA MAY BE SELECTED AND INSERTED FROM THE ADDRESSFILE (B) AS WELL AS FROM THE MASTER FILE 2 (C).

FILEACCESS (5)

```
Form 4                    Businessletters                          IDT 3000

Text name [        ]    Section [   ]    Password [     ]

[ ] (1) From unformatted text or (2) formatted file  to printer No. [ ]

[ ] (1) A letter to addressee:                                   (B)
    (2)                 without accessing the address-file
(A) (3) A letter to all addresses of the address-file
    (4) Only selected addresses with field __ _ ____________________
                                  [   ] field __ _ ____________________

[ ] (X) Using letterhead:  ____      dated for renewed submission : _____

                                                                 (C)
_   (X) With data of  Order   -  Master-file of  ____________________
_   (X) Margin alignment              ___ Column (only printer 2601)

Your sign        Your letter                Our sign       Date
_____________________________________________________    _____________

Please enter the required processing/selection options and press -ENTER- !
```

THE FILEACCESS (AND THE DATA SELECTION) CAN BE DONE DURING THE PRINTING / SELECTING OF BUSINESS LETTERS (A). DATA MAY BE SELECTED AND INSERTED FROM THE ADDRESSFILE (B) AS WELL AS FROM THE MASTER FILE 2 (C).

DATA SELECTION (1)

THE SELECTION OF DATA OUT OF THE ABOVE MENTIONED DATA FILES CAN BE DONE IN THE FOLLOWING WAYS:

- THROUGH LOGICAL COMPARE COMMANDS DURING THE BUSINESS LETTER PRINTING (SELECTED LETTERS)

- THROUGH DATA(BASE) INQUIRY LANGUAGES AND/OR REPORT GENERATORS (QUERY, REPORT3000, ASK, QUIZ, QUICK, GENEASYS ETC.) PROVIDING THE SELECTED DATA IN A IDT3000 BATCH INTERFACE FILE

- THROUGH BATCH- OR SESSION MODE APPLICATION PROGRAMS PROVIDING THE SELECTED DATA IN AN IDT3000 BATCH INTERFACE FILE

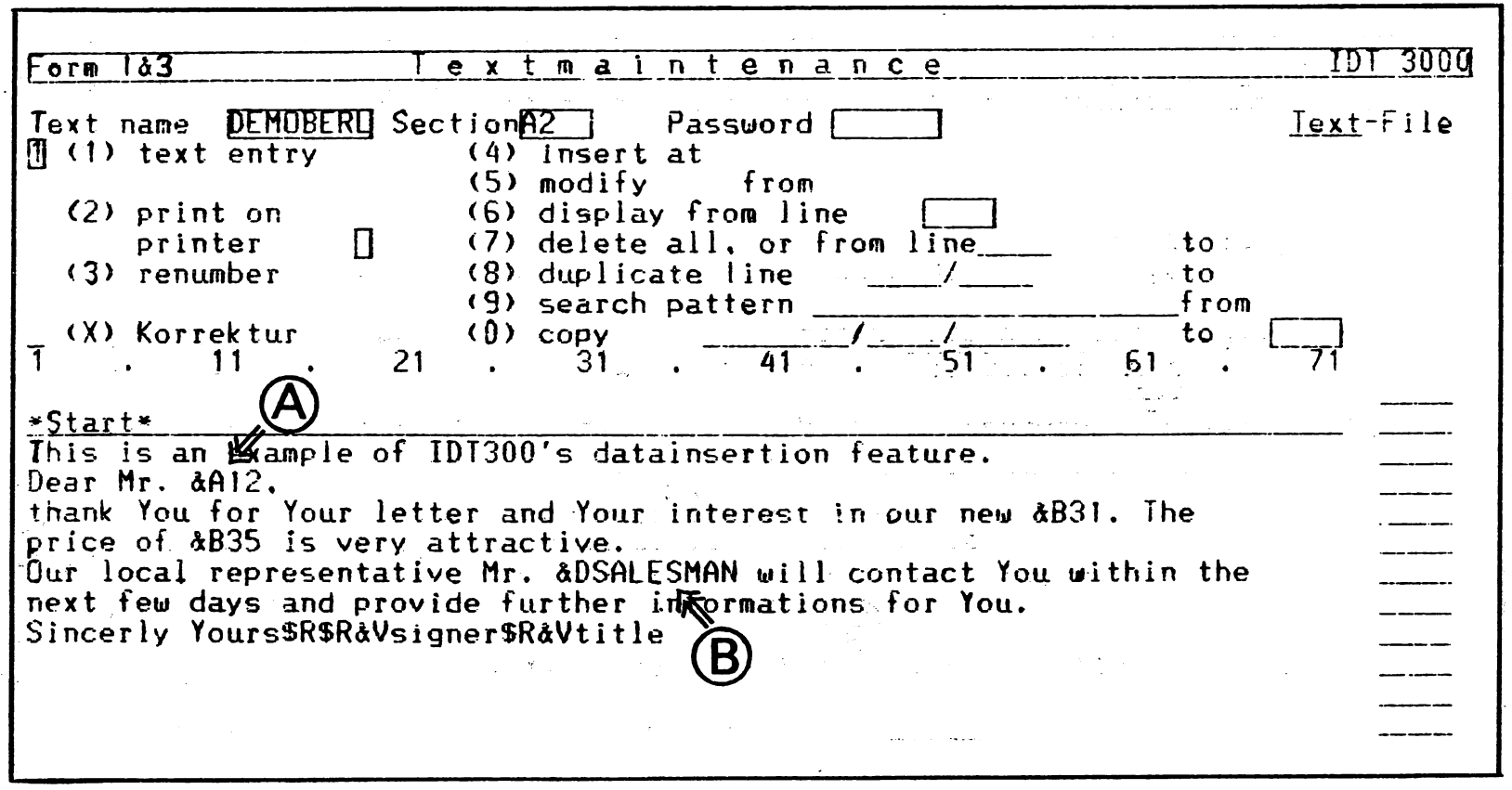
SEE EXAMPLES NEXT PAGES ...

DATA SELECTION (2)

```
Form 4                    Businessletters                          IDT 3000

Text name DEMOBERU    Section A1    Password [    ]

2  (1) From unformatted text or (2) formatted file  to printer No. 3

1  (1) A letter to addressee:        1000008 (B)
   (2)                  without accessing the address-file
(A)(3) A letter to all addresses of the address-file
   (4) Only selected addresses with field __ = ____________________
                                [    ] field __ = ____________________

X  (X) Using letterhead:    std1      dated for renewed submission : 020381

X  (X) With data of  Order  -  Master-file of  935-001/21C (C)
_  (X) Margin alignment                ___ Column (only printer 2601)

Your sign        Your letter                 Our sign        Date
                                                             5. 8. 1981

Please enter the required processing/selection options and press -ENTER- !
```

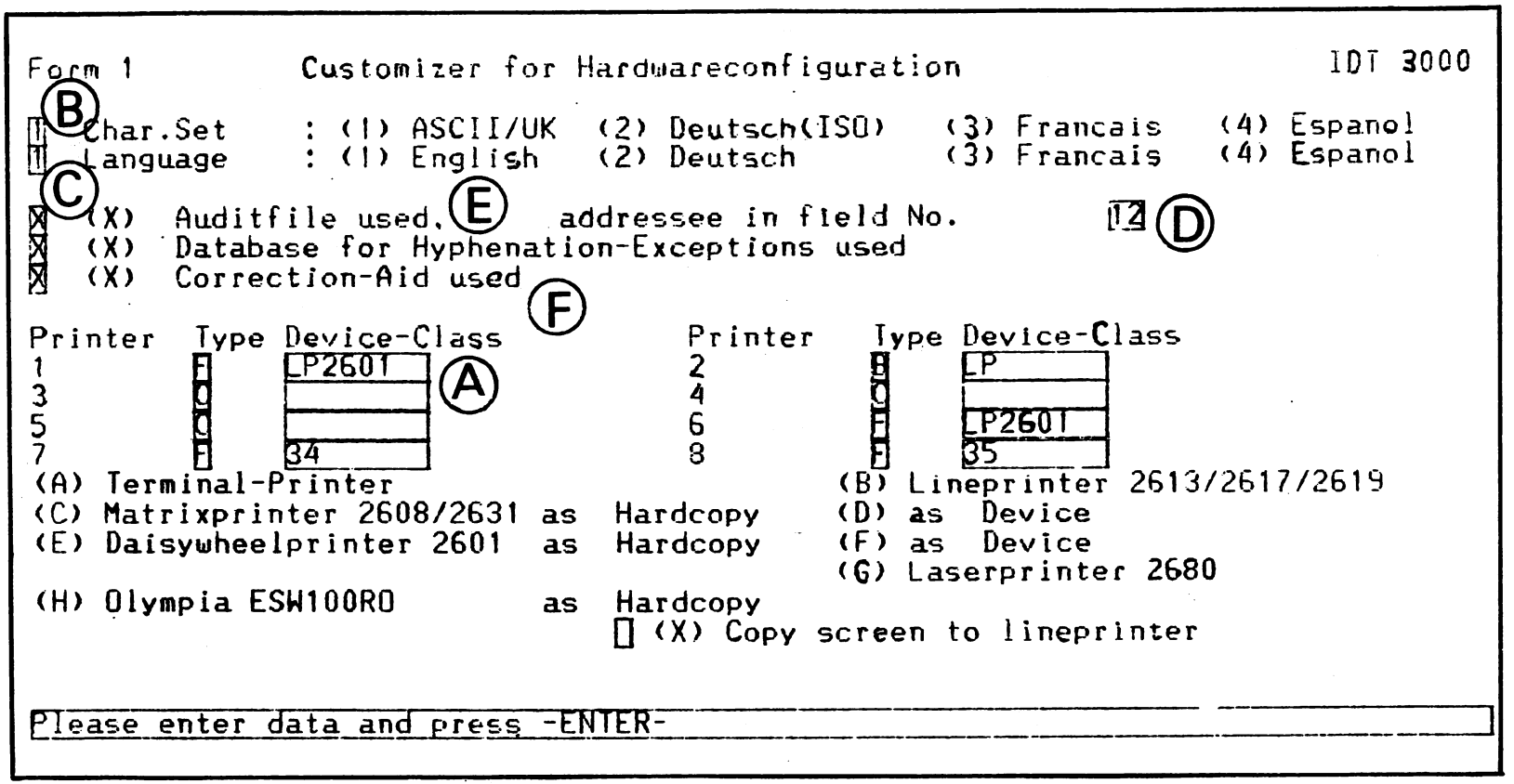
THE SELECTION OF DATA CAN BE DONE IN THE TEXT FORMATTING MODULE OR DURING THE PRINTING OF SELECTED BUSINESS LETTERS (A) WITH DATA ELEMENTS OF THE SPECIFIED ITEMS (B) + (C).

DATA SELECTION (3)



DATA ELEMENTS CAN BE DEFINED BY IDT3000 INTERNAL FIELD NUMBERS (A) (REFERRING TO THE CUSTOMIZED FILE-ENVIRONMENT) OR BY DATA ELEMENT NAMES OF A DATA DICTIONARY (B) (DICTIONARY 3000).

CUSTOMIZER (1)



THE INTEGRATION OF DATA- AND TEXTPROCESSING REQUIRES THE OPTION OF TEXTPROCESSING WITHIN THE EXISTING DATAPROCESSING (HARDWARE-)ENVIRONMENT (A) INCLUDING CHARACTER SETS (B), LANGUAGES (C), DATA LAYOUTS (D) AND THE PROCESSING ENVIRONMENT (E) + (F).

SUPERVISOR

Form 1 — Daily Report/Renewed submission — IDT 3000

[ ] (1) Print daily report (A) — Printer No. [ ]
(2) daily report only for business letters
(3) check list for renewed submission (B)

[ ] (A) Totally
(B) Date of beginning
(C) From this date up to (C)

Sort item (D)
1st [ ]
2nd _
3rd _
4th _

(A) Print date
(B) Date of renewed submission
(C) User
(D) Text name
(E) Adressee

Optional
only 'text name' _____
only user ____

Comment : _____

THE SUPERVISOR FEATURE ALLOWS SOME KIND OF PROCESSING FUNCTIONS LIKE DAILY REPORTS OF OUTGOING LETTERS AND REPORTS (A) OR RECORDS OF RENEWED SUBMISSIONS, THAT ARE DUE FOR NEW ACTIONS (B). BOTH REPORTS MAY BE TOTAL OR A PARTIAL SELECTION (C) WITH DIFFERENT SORT CRITERIA (D).

INTERFACES

INTEGRATED DATA- AND TEXTPROCESSING REQUIRES SEVERAL INTERFACES BETWEEN THE TEXTPROCESSING AND

- THE COMPUTER FILES (MPE, KSAM, IMAGE)
- DATA DEFINITIONS (TRADITIONAL FILE LAYOUTS, DATA BASE SCHEMAS, DATA DICTIONARIES)
- THE BATCH PROCESSING
- DATA SELECTIONS OUT OF QUERY, ASK, REPORT, APPLICATION PROGRAMS ETC.
- VARIOUS TYPES OF PRINTING HARDWARE (FROM TYPEWRITER TO LASER PRINTER)
- DATA EXCHANGE AND DATA COMMUNICATION (MAGNETIC TAPES, MICROFICHES, POINT TO POINT DATA COMMUNICATION, MULTIPOINT DATA-COMMUNICATION, COMPUTER-COMPUTER COMMUNICATION, ELECTRONIC MAIL ...)

Computerized Typesetting: TEX on the HP3000

Lance Carnes
Independent Consultant
Mill Valley, California

ABSTRACT

TEX is a program which allows the ordinary user to produce professional quality typeset output. TEX was developed by Donald E. Knuth of Stanford University and is currently used throughout the world for typesetting both technical and non-technical material. This paper will describe the use of TEX and show some examples of its output. The transportable version of TEX, written in PASCAL, has been successfully moved to the HP3000. The second part of the paper describes the tasks involved in this process.

INTRODUCTION

1. What is TEX?

Tau Epsilon Chi (TEX) is a system for typesetting technical books and papers. It can also be used for ordinary non-technical material. The system does not require the user to have a knowledge of typesetting rules or conventions.

The original TEX system was developed at Stanford University by Donald E. Knuth. Frustrated in his attempts to print a second edition of *The Art of Computer Programming* in the same printing style as the first edition, he looked for alternatives in the area of computerized typesetting. Finding nothing that suited him, he embarked on a project which was to become the TEX system. This system is described in detail in his informative and humorous book, *TEX and METAFONT* [Knut79].

The TEX system is currently used throughout the world, partly for technical work in mathematics and physics, and partly for various other uses. The *Journal of the American Mathematical Society* now accepts TEX input files for publication. Some major corporations and universities use it for typesetting their internal documentation, user manuals, newsletters, etcs. The TEX Users Group accepts articles and letters for their Journal in TEX format.

2. How Does It Work?

The TEX program accepts an input file consisting of text and control sequences, and generates a device independent output file (DVI file) which contains commands for driving a raster printer device. Once TEX has processed the input and produced a DVI file, it is up to a device driver program to interpret the commands in the DVI file and produce printed output. This sequence of events is shown in Figure 1.

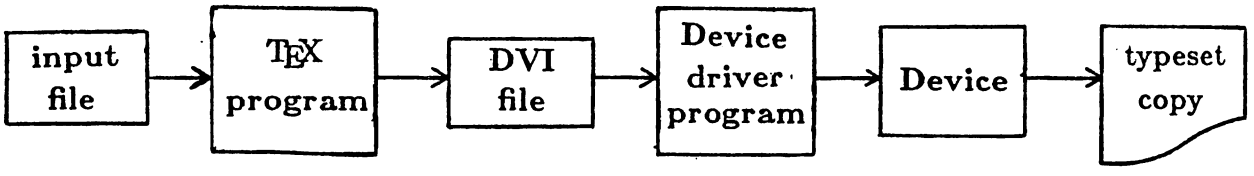


Figure 1. Functional Diagram of the TEX System

Most of the typesetting is done by TEX automatically. TEX operates on many levels, composing pages, paragraphs lines and words. All of these are interrelated, with the intention of producing professional quality printing. In cases where TEX needs to be guided, for example in printing the TEX logo, the user intervenes by specifying a control sequence (see 3 below).

The TEX system does not typeset a single word or a single line at a time. Rather, it typesets a page or more at a time. This is done for a variety of reasons. Mainly, we want the printed page to consist of pleasantly spaced paragraphs, lines and words. Also we want to avoid other unwanted phenomena, such as "widow" lines. A widow line is the first line of a paragraph appearing at the bottom of a page with the paragraph continuing on the next page. To eliminate widows, TEX returns to the paragraphs already layed out and expands them slightly so as to use one more line on the page. This forces the widow line to the top of the next page.

Paragraphs are composed to reduce the number of

hyphenations and so as not to leave a single word stranded in the last line. In addition, the spacing between words is equalized throughout the paragraph.

Lines of text are composed of words and other symbols (e.g., mathematical formulas) with the space between words equalized.

Words are typeset with the letters placed one character width apart. Unlike standard computer printers which print all characters in the same width (usually 1/10 inch), typesetting separates characters by the exact width of the character, depending on the "font" or character style used. In addition, TEX will place characters closer together or farther apart in accordance with traditional typesetting rules. For example, when typesetting the word "AVIATOR" the "A" and "V" are placed closer together; this is called "kerning." Notice in the word "find" that the "f" and "i" are pushed together to form the "ligature" fi. These typesetting conventions and more are known to TEX, freeing the user from having to memorize them.

The basic concepts TEX uses are "boxes" and "glue." A box contains something which is to be printed, and glue specifies the spacing between boxes. For example, a character is a box, a word is a collection of character boxes, a line is a group of word boxes, a paragraph is a collection of line boxes, and a page is a box composed of paragraph boxes. The space between boxes can expand or contract by carefully defined amounts, called the stretchability or shrinkability of the glue. For example, when TEX composes a paragraph that has a hyphenation it tries to back up and redistribute the spacing of the words in the paragraph to avoid the hyphenation. It does this by increasing or shrinking the space or glue between the boxes by allowable amounts.

For further details on the inner workings of TEX, see [Knut79] or [Spiv80].

3. Submitting an Input File to TEX

The input file for TEX is edited using any text editor. The text and any control sequences are contained in this file. When TEX is run, this file is designated as the input file.

Basically, text is entered in a standard fashion with spaces between words, and one blank line between paragraphs. The input need not be formatted in any particular manner beyond this. Control sequences are defined as a "\" followed by a word or symbol. They allow the user to specify a special command. For example, "\it IMPORTANT" would cause the world IMPORTANT to be set in italic font.

The TEX system can be run in either interactive or batch mode. In interactive mode, if TEX finds an error the user is allowed to make modifications on the fly. For example,

```
!Undefined control sequence
\iy
```

IMPORTANT

The TEX program is indicating that it does not know the control sequence "\iy" and shows what it has scanned on the first line, and what it has not yet scanned on the following line. At this point the user may correct the input by typing "1" to erase one symbol or control sequence, and then "I" to insert the correct sequence "\it". Any corrections made in this manner are recorded in an errors file for future reference.

As TEX is processing the input, it is writing to the DVI file. After the input is successfully processed, the DVI file is ready for the device driver program.

Two other important facilities are available with TEX. These are alternate input files and macro definitions. Alternate input files are TEX input files which are read in conjunction with another input file. For example, if a paper has an abstract and three sections, and each is in a separate file, a main file would draw them all together as follows:

```
% paper on TEX for the HP3000
\input basic % basic control sequences
\input texabs % abstract file
\input sect1
\input sect2
\input sect3
\end
```

Each of the alternate input files could have had \input commands also. The maximum nesting depth is nine.

Macro definitions allow the user to specify a common sequence by defining it and giving it a name. For example, the logo TEX was specified by inserting "\TEX". \TEX was previously defined as

```
\def TEX{\hbox{lowercase{\:a
\uppercase{T} hskip-2pt\lower1.94pt
\hbox{\uppercase{E}}\hskip-2pt \uppercase{X}}}}
```

It is much easier to write "\TEX" than to insert the above expansion.

4. Fonts

A font is a specific design of an alphabet and associated symbols. Most typewrites have Pica or Elite type fonts. The different "balls" or "daisy wheels" on some printers allow the user to change fonts.

The TEX system allows up to 64 different fonts to be specified within the same job. A control sequence is given to switch from one to the other. Naturally you must have a device which can support all of these different fonts.

Knuth also wanted to define his own fonts and created a system called METAFONT to do this. Using METAFONT one can design a font which is coded into a file for use by TEX. For more information on METAFONT see [Knut79].

5. The DVI File

The DVI file consists of a series of 8-bit codes which

tells a device driver how to typeset the job. The format of the DVI files is given in Appendix B.

Basically, a DVI file command is of the form "set the letter d and advance the character width" or "change to font 3" or "advance vertically 12 rsu's". No inherent intelligence on the part of the device is assumed. In fact, TEX gets along best with devices which have no internal programming, such as proportional spacing or typesetting firmware.

6. Device drivers.

The assumed printer is a raster scan printing device. This implies that all spacing between characters and lines is user specified. A typical computer line printer is not a raster device since it will always print 10 character/inch, and six lines/inch (or some variation of this). Most of the daisy wheel terminals available now can be used as raster devices. The actual device TEX is aimed at is a commercial computer-driven typesetting device, such as a Xerox Graphics Printer, a Mergenthaler Linotron 202, or an HP2680 Laser Printer.

The TEX program has no knowledge of any particular printing device. It creates the same DVI file regardless of the output device. It is the job of the Device Driver program to interpret the DVI commands and produce output on a specific device. While there is only one TEX program, there will be one Device Driver program for each output device.

TEX ON THE HP3000

1. TEX in PASCAL

The TEX system was originally written in a language called SAIL (Stanford University Artificial Intelligence Language). The SAIL compiler and the original TEX system ran on the DEC-20 computer only. TEX is in the public domain, but was not even remotely transportable. Due to the popularity of the system, a project was undertaken to translate the TEX system into a computer language which was available on most modern computer systems.

The language chosen for the transportable system was PASCAL. The method for translating the system was as follows. First, a well documented pseudo-PASCAL source was developed. This source has only a slight resemblance to a PASCAL program and was intended to serve mostly as documentation, and to give all the algorithms. This file is often referred to as the DOC file.

The second step was to produce syntactically correct PASCAL source code from the DOC file. There is a program called UNDOC which performs this step. The resulting PASCAL source is distributed to anyone wanting to transport TEX to another computer.

The DOC file is actually typeset, and a photocopy is provided with the distribution tape. The PASCAL source is almost unreadable, but will compile. Examples of both of these files is in Appendix C.

2. Moving TEX to the HP3000

The Stanford TEX-in-PASCAL project brought the system to a point where it could be transported to other computer systems. The transportation process, however, requires a good deal of time and a patient systems programmer.

At the time of writing, this author has successfully transported TEX to the HP3000. The project was by no means trivial, as will be shown.

Bringing TEX to the HP3000 had a lot of problems right from the outset. First, there was no supported PASCAL compiler at the time this project was begun. Second, the design of the TEX program assumes a large address space, something on the order of 600K words of addressable memory.

The tasks broke down as follows:

a. Edit the PASCAL sources. While the system was translated to a "Standard" PASCAL, there are still many variations and assumed extensions which had to be accounted for. With 23,000 lines of PASCAL source this took considerable time and effort.

b. Rewrite the "System dependent" routines. These are the procedures and functions which interface TEX with the file system, terminal I/O and other traits particular to the host system. About 25 routines had to be modified or rewritten.

c. Implement a virtual memory scheme. TEX references several large arrays throughout, some as large as 50,000 elements with 4 32-bit words per element. An addressing scheme was developed to allow the array contents to reside in secondary storage.

d. Revise the PASCAL compiler to allow 32-bit integers and to compile large array references. TEX assumes 32-bit integers throughout, and the Portable P4 compiler from the HP Users Contributed Library was modified to allow them.

e. Optimize the performance of the system. When the above tasks were completed and the system first ran on the HP3000, it was incredibly slow. Where the original TEX system at Stanford processed a document in less than two minutes, the initial HP3000 TEX took 40 minutes. By analyzing TEX's operation, some optimizations have been made reducing the run time to about 6 minutes. Additional optimizations will be made to allow the system to run as fast as possible. One tool which has been particularly useful for identifying inefficient code is APG/3000 from Wick Hill Associates.

3. Device drivers

A device driver for a daisy wheel printer has been developed for use on the HP3000. While only one font is available at a time with this device, satisfactory results have been obtained. The output is suitable for internal documentation, and for proofing a document. Future plans are to develop a driver for the HP2680 Laser printer.

However, it is not necessary to have a high quality

printing device on-site. There is one commercial printing house in San Francisco which uses TEX for typesetting on a Mergenthaler Linotron 202; the output from this device is camera ready. DVI files produced by TEX on the HP3000, once proofed on the daisy wheel printer, will be sent to this commercial printer.

CONCLUSIONS

This is a truly remarkable system. It gives the ordinary person the ability to print professional quality copy. The user will not have to explain to a typographer what is wanted, but will have personal control.

The HP3000 implementation of TEX will be a boon for any organization desiring to improve the quality of documentation, user manuals and other printed materials. Good results can be obtained with an inexpensive daisy wheel printer. Where camera-ready copy is desired, several higher quality devices are commercially available.

Hopefully more organizations will begin to use TEX for documentation, manuals, annual reports and newsletters. Perhaps one day soon the HP General Systems Users Group will accept papers for publication in TEX format.

ACKNOWLEDGEMENT

My thanks to Prof. Luis Trabb-Pardo and Charles Restivo of Stanford University for their assistance in learning the TEX system; and to GENTRY, INC. of Oakland, California, for providing time on the HP3000.

REFERENCES

[Knut79] Donald E. Knuth, *TEX and METAFONT*. New Directions in Typesetting. Digital Press, 1979.

This is a beautifully printed book, an acknowledgement of the TEX system. Don Knuth's writing style is at once brilliant and witty. It contains a User's Guide to the TEX and METAFONT systems and a paper on Mathematical Typography.

[Spiv80] Michael Spivak, *The Joy of TEX*. A Gourmet Guide to Typesetting Technical Text by Computer. Verson -1. American Mathematical Society, 1980.

This is a real book. It gives a lighthearted introduction to the use of AMS-TEX, the version of TEX used by the AMS.

TUGboat, The TEX Users Group Newsletter. Published by the American Mathematical Society.

The TEX Users Group is small currently, but enthusiastic and helpful. For information on membership write to:

TEX Users Group
c/o American Mathematical Society
P.O. Box 6248
Providence, Rhode Island 02940

```
\noindent {\bf ABSTRACT:} \TEX\ is a program which allows
the ordinary user to produce professional quality
typeset output.
\TEX\ was developed by Donald E. Knuth of Stanford
University and is currently used throughout the world
for typesetting both technical and non-technical material.
This paper will describe the use of \TEX\ and show
some examples of its output.
The transportable version of TEX, written in Pascal,
has been successfully moved to the HP3000.
The second part of the paper describes the tasks involved
in this process.

\vskip 0.4 cm
\noindent {\bf I. INTRODUCTION}

\vskip 0.3 cm
\noindent {\bf 1. What is \TEX\ ?}

\vskip 0.1 cm
{\it Tau Epsilon Chi} (\TEX\ ) is a system for typesetting
technical books an dpapers.
It can also be used for ordinary non-technical material.
The system does not require the user to have a knowledge of
typesetting rules or conventions.

The original \TEX\ system was developed at Stanford University
by Donald E. Knuth.
Frustrated in his attempts to print a second edition of
{\it The Art of Computer Programming} in the same printing
style as the first edition, he looked for alternatives in the
area of computerized typesetting.
Finding nothing that suited him, he embarked on a project
```

ABSTRACT: TEX is a program which allows the ordinary user to produce professional quality typeset output. TEX was developed by Donald E. Knuth of Stanford University and is currently used throughout the world for typesetting both technical and non-technical material. This paper will describe the use of TEX and show some examples of its output. The transportable version of TEX, written in Pascal, has been successfully moved to the HP3000. The second part of the paper describes the tasks involved in this process.

I. INTRODUCTION

1. What is TEX ?

Tau Epsilon Chi (TEX) is a system for typesetting technical books and papers. It can also be used for ordinary non-technical material. The system does not require the user to have a knowledge of typesetting rules or conventions.

The original TEX system was developed at Stanford University by Donald E. Knuth. Frustrated in his attempts to print a second edition of *The Art of Computer Programming* in the same printing style as the first edition, he looked for alternatives in the area of computerized typesetting. Finding nothing that suited him, he embarked on a project

APPENDIX A
A PORTION OF THE TEX INPUT FILE

Command Name	Command Bytes Description
VERTCHAR0	0 Set character number 0 from the current font such that its reference point is at the current position on the page, and then increment horizontal coordinate by the character's width.
VERTCHAR1	1 Set character number 1, etc.
⋮	⋮
VERTCHAR127	127 Set character number 127, etc.
NOP	128 No-op, do nothing, ignore. Note that NOPs come *between* commands, they may not come between a command and its parameters, or between two parameters.
BOP	129 c0[4] c1[4] ... c9[4] p[4] Beginning of page. The parameter p is a pointer to the BOP command of the *previous* page in the .DVI file (where the *first* BOP in a .DVI file has a p of −1, by convention). The ten c's hold the values of TEX's ten \counters at the time this page was output.
EOP	130 The end of all commands for the page has been reached. The number of PUSH commands on this page should equal the number of POPs.
PUSH	132 Push the current values of horizontal coordinate and vertical coordinate, and the current w-, x-, y-, and z-amounts onto the stack, but don't alter them (so an X0 after a PUSH will get to the same spot that it would have had it had been given just before the PUSH).
POP	133 Pop the z-, y-, x-, and w-amounts, and vertical coordinate and horizontal coordinate off the stack. At no point in a .DVI file will there have been more POPs than PUSHes.
HORZRULE	135 h[4] w[4] Typeset a rule of height h and width w, with its bottom left corner at the current position on the page. If either $h \leq 0$ or $w \leq 0$, no rule should be set.

APPENDIX B
DVI COMMANDS

VERTRULE — 134 h[4] w[4]
Same as HORZRULE, but also increment horizontal coordinate by w when done (even if h $\leq$ 0 or w $\leq$ 0).

HORZCHAR — 136 c[1]
Set character c just as if we'd gotten the VERTCHARc command, but don't change the current position on the page. Note that c must be in the range [0..127].

FONT — 137 f[4]
Set current font to f. Note that this command is not currently used by TEX—it is only needed if f is greater than 63, because of the FONTNUM commands below. Large font numbers are intended for use with oriental alphabets and for (possibly large) illustrations that are to appear in a document; the maximum legal number is $2^{32} - 2$.

X2 — 144 m[2]
Move right m rsu's by adding m to horizontal coordinate, and put m into x-amount. Note that m is in 2's complement, so this could actually be a move to the left.

X3 — 143 m[3]
Same as X2 (but has a 3 byte long m parameter).

X4 — 142 m[4]
Same as X2 (but has a 4 byte long m parameter).

X0 — 145
Move right x-amount (which can be negative, etc).

W2 — 140 m[2]
The same as the X2 command (i.e., alters horizontal coordinate), but alter w-amount rather than x-amount, so that doing a W0 command can have different results than doing an X0 command.

W3 — 139 m[3]
As above.

W4 — 138 m[4]
As above.

W0 — 141
Move right w-amount.

Y2 — 148 n[2]
Same idea, but now it's "down" rather than "right", so vertical coordinate changes, as does y-amount.

Y3	147 n[3] As above.
Y4	146 n[4] As above.
Y0	149 Guess.
Z2	152 m[2] Another downer. Affects vertical coordinate and z-amount.
Z3	151 m[3]
Z4	150 m[4]
Z0	153 Guess again.
FONTNUM0	154 Set current font to 0.
FONTNUM1	155 Set current font to 1.
⋮	⋮
FONTNUM63	217 Set current font to 63.

36. The procedure *Print* takes an integer as argument and prints the corresponding *strngpool* entry both in the terminal and in the errors file.

```
procedure Print(mes : integer);
  var i : integer; { index in the string }
    c : asciiCode;
  begin i := strng[mes]; c := strngpool[i];
  while c <> null do
    begin terOut↑ := chr(c); errfil↑ := chr(c); put(terOut); put(errfil);
    Increment(i); c := strngpool[i]
    end;
  end;
procedure PrintLn(mes : integer);
        { Like Print, but beginning at a new line. }
  begin terOut↑ := chr(carriagereturn); errfil↑ := terOut↑; put(terOut);
  put(errfil); terOut↑ := chr(linefeed); errfil↑ := terOut↑; put(terOut);
  put(errfil); Print(mes)
  end;
```

```
1224 PROCEDURE PRINT(MES: INTEGER);
1224 VAR I: INTEGER;
   3 S: ASCIICODE;
   5 B
1224 I:=
1233 C:=ST
1247 BEGIN
1247 TEROUT^:= HR( );
1251 ER FIL^:= HR( );
1255 PUT(T P UT);
1257 PUT (ERPFIL);
1259 I:=I+1;
1263 C:=STRNGPOOL[1]
1270 END;
1274 END;
1275 PROCEDURE PRINTLN(MES: INTEGER);
1275 BEGIN
1275 PRINT(MES)
1279 ;WRITELN(TEROUT);
1282 WRITELN(ERRFIL);
1284 END;
```

APPENDIX C
FRAGMENTS OF TEX DOC AND PASCAL SOURCE

25. Find the equation of the plane passing through the end points of the three vectors $\mathbf{A} = 3\mathbf{i} - \mathbf{j} + \mathbf{k}$, $\mathbf{B} = \mathbf{i} + 2\mathbf{j} - \mathbf{k}$, and $\mathbf{C} = \mathbf{i} + \mathbf{j} + \mathbf{k}$, supposed to be drawn from the origin.

26. Show that the plane through the three points (x_1, y_1, z_1), (x_2, y_2, z_2), and (x_3, y_3, z_3) is given by

$$\begin{vmatrix} x_1 - x & y_1 - y & z_1 - z \\ x_2 - x & y_2 - y & z_2 - z \\ x_3 - x & y_3 - y & z_3 - z \end{vmatrix} = 0.$$

Solution. Let $w = f(x, y, z) = x^2 + y^2 - z$, so that the equation of the surface has the form

$$f(x, y, z) = \text{constant},$$

Everything You Wanted to Know About Interfacing to the HP3000 PART I

Ross Scroggs
The Type Ahead Engine Company
Oakland, California

INTRODUCTION

It is important to realize that the information presented in this paper is my interpretation of the facts. The interpretation is not perfect, for surely I have included incorrect statements. If you believe that something here is incorrect, bring it to my attention. If I believe that you are wrong I will try to set you straight, but I will not argue about anything. I have included a list of references at the end of this paper from which I have obtained most 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 HP3000 and don't believe anything you read, if you haven't seen it work yourself, plan on having to solve a few problems. This paper is a guide to solving those problems, but it won't solve them for you.

Most of the experiments outlined in this paper were performed with the Bruno release of MPE-IV, I have subsequently been informed that the C release of MPE-IV fixed many terminal driver problems associated with the ADCC.

Asynchronous terminals are attached to the HP3000 Series I, II, and III through the Asynchronous Terminal Controller (ATC) and to the Series 30, 33, 40, and 44 through the Asynchronous Data Communications Controller (ADCC). This paper addresses issues involved in making a successful connection to one of these two devices. Terminals attach to the Series 64 through the Advanced Terminal Processor (ATP) which should make all of our lives simpler (though expensive) in the coming years. In its earlier versions the ATP will act much like the ATC in terms of interfacing to terminals. It features two major advances over the previous terminal controllers. First, there is a microprocessor controlling each terminal line, this removes considerable work from the CPU, the "character interrupt" problem. Second, the ATP can use either the RS-232 or RS-422 interface standards. RS-422 is a completely new electrical and mechanical interface that supports very high data rates over great distances with no errors, a typical example would be 9600 baud at 4000 feet. What this flexibility costs you is about $200 extra per terminal to provide a RS-232 to RS-422 adapter. These won't be required when terminals provide RS-422 interfaces.

Terminals attached to the ATC or ADCC are accessed primarily 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 HP3000 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 and 9600 baud. Unfortunately these two higher speeds can not be sensed by the ADCC and thus you must log on at a lower speed and use the MPE SPEED command to access the higher speed. (Use of subtype 4 and specifying any speed will allow a terminal to log on at that speed only, this includes 4800 and 9600. Note however, that if you use the :SPEED command the new speed specified will be required at your next logon.)

Terminal Parity

The format of characters processed by the HP3000 is a single start bit, seven data bits, a parity bit, and one stop bit (two at 110 baud). The parity bit may always be zero, always be one, computed for odd parity, or computed for even parity. Choosing the proper parity setting has been complicated by differences between the ATC and ADCC. 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 requesting program. The ADCC also sets parity based on the parity bit of the initial carriage return but does so with a slight, but nasty twist. If the bit is a zero the ADCC passes through the parity bit supplied by the application program on output, if it is a one the ADCC generates even parity on output. If pass through parity was selected the parity of the incoming data is passed through to your program buffer. If even parity was selected the input data is checked for proper even parity. Thus, you should not use odd or force to one parity on the ADCC. The odd parity will be interpreted as pass through and the parity bits will wind up in your data buffer, string comparisons will fail because of the parity bits. Force to one parity will be interpreted as even and all input will cause parity errors.

Subtype

The ATC supports subtypes 0, 1, 2, 3, 4, 5, 6, 7, the ADCC support subtypes 0, 1, 2, 3, 4, 5. Subtypes 2, 3, 6, 7 concern half duplex modems and not me, so I will ignore them. Subtype 0 is the standard for directly attaching terminals without modems. (Note that terminals that are attached to multiplexors can fit in this category, the modem involved is managed by the multiplexor, not the HP3000.) Subtype 1 is the standard for attaching terminals that use full duplex modems such as Bell 103, 212 and Vadic 34xx. Both subtypes 0 and 1 speed sense on the initial carriage return. Subtype 4 is for direct attach terminals that will not be speed sensed, they will run at a fixed speed that is set at configuration time. This subtype is often used to prevent the HP3000 from trying to speed sense garbage, this sometimes occurs when using short-haul modems (line-drivers) that do not have a terminal attached to the other end. Subtype 5 is for modem attached terminals that will not be speed sensed.

Termtype

The ATC supports terminal types 0, 1, 2, 3, 4, 5, 6, 9, 10, 12, 13, 15, 16, 18, 19, 31, the ADCC supports terminal types 4, 6, 9, 10, 12, 13, 15, 16, 18, 19. 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, truly bizarre. (Termtype 4 on the ADCC does not output DC3s at the end of each line.) Termtype 6 is for low speed printers, 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 purpose non-HP CRT terminal type. No DC3s are output at the end of the line (whew!!) and nothing strange happens on backspace, the cursor backs up just as you would expect. Termtype 10 is the standard for HP-26xx terminals. Termtype 13 is typically for those terminals at a great distance from the HP3000 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 HP3000 echo them.) Termtypes 15 and 16 are for HP-263x printers. Termtype 18 is just like termtype 13 except that no DC1 is issued on a terminal read. Certain termtypes less than 10 specify a delay after carriage control characters are output to the terminal. The ATC handles this by delaying for the designated number of character times but does 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.

Cable

Direct attach 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 HP3000 which acts like a modem.) Typically the cable will connect pin 2 at the terminal end to pin 2 at the HP3000, pin 3 at the terminal to pin 3 at the HP3000 and pin 7 at the terminal to pin 7 at the HP3000. This is not to say that your terminal does not require other signals, it just says that the HP3000 is not going to provide them, you must. 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 attach 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 HP3000 is acting like a modem and it is being attached to a modem. Typically, the cable that connects the HP3000 to the modem will connect pin 2 at the modem end to pin 3 at the HP3000, pin 3 at the modem to pin 2 at the HP3000, pin 4 at the modem to pin 8 at the HP3000, pin 6 at the modem to pin 20 at the HP3000, pin 7 at the modem to pin 7 at the HP3000, pin 8 at at the modem to pin 4 at the HP3000, and pin 20 at the modem to pin 6 at the HP3000.

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 speed/amount of data from the HP3000 to the terminal is con-

trolled. The HP3000 supports two flow control methods, ENQ/ACK and XON/XOFF. The ENQ/ACK protocol is controlled by the system, after every 80 output characters the systems 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 that attempt to access the HP3000 through a port that is configured for an HP terminal. Most terminals do not respond to an ENQ with an ACK, you must do it manually by typing Control-F which is an ACK. An ENQ is generated by the HP3000 when the initial carriage return is received from the terminal, thus you get hung immediately. But, hit Control-F, and logon and specify the proper termtype in your HELLO command.)

The XON/XOFF flow control protocol is controlled by the terminal. When the terminal wishes to suspend output from the HP3000 it sends an XOFF (Control-S or DC3) to the HP3000 and sends an XON (Control-Q or DC1) to resume output. Unfortunately the HP3000 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 HP3000 misses the XOFF. Termtype 13 has in my experience been the best termtype to use if your terminal requires the XON/XOFF flow control protocol. You can turn the echo back on with ESC :.

A special note on XON. If you inadvertently send an XON (DC1) to the HP3000 when output is not suspended, surprise you are now in paper tape mode and backspace, Control-X, and linefeed will act most strangely. Hit a single Control-Y to get out of this mode, the Control-Y will not be received by your program.

Some terminals perform flow control by raising and lowering a signal on their interface, the HP3000 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, a costly affair.

A form of flow control used by HP terminals when inputting data to the HP3000 is the DC2/DC1 protocol. When the enter key is pressed on the terminal, a DC2 is sent to the HP3000 to alert it to a pending block mode transfer. When the HP3000 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 HP actually sends DC2 carriage return when the enter key is pressed. This is no problem unless the DC2 and CR do not arrive 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.

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 HP3000. 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 and will give short (incomplete) program segments that illustrate the access method.

Declarations

The following declarations will be assumed for all program segments shown.

```
begin

integer
    ilen,
    olen,
    pfnum:=0,
    pifnum:=0,
    pofnum:=0,
    precsize:=-256;  <<** pick a number large enough for the
                        maximum data transfer **>>

logical
    fcontrol'parm:=0,
    prev'echo;

logical array
    ibuff'(0:255),
    obuff'(0:255),
```

```
byte array
    pfname(0:7):="PROGDEV ";

byte array
    pdevice(0:7):="PROGDEV ";

byte array
    pifname(0:7):="IPROGDV ";

byte array
    pidevice(0:7):="IPROGDV ";

byte array
    pofname(0:7):="OPROGDV ";

byte array
    podevice(0:7):="OPROGDV ";

define
    fs'error'on'ccl= if <  then file'error(#,
    fs'error'on'ccne=if <> then file'error(#;

procedure print'message(enum);
  value
    enum;
  integer
    enum;
  option external;

intrinsic
    fclose,fcontrol,fopen,fread,fsetmode,fwrite,print'file'info,
    getprivmode,getusermode,iowait,terminate;

subroutine file'error(fnum,enum);
  value
    fnum,enum;
  integer
    fnum,enum;
  begin

    <<** simple file error handling subroutine, basic, not fancy
         or very good. **>>

    print'file'info(fnum);
    print'message(enum);  <<** supply something, but remember your
                              cliches, make it user-friendly! **>>
    terminate  <<** simple, direct, not too graceful **>>
  end;  <<* file'error *>>
```

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 exclusive access and input/output. Choose a record size that is larger than the maximum data transfer that will take place.

ATC — Opening a terminal with an HP termtype causes an initial ENQ to be output to the device on the first output, there must be an ACK reply from the device or your program will wait until the ENQ time-out occurs.

ADCC

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 speed sensed by the HP3000.

```
    pfnum:=fopen(pfname,%604,%104,precsize,pdevice);
    fs'error'on'ccl(pfnum,1);
```

FCLOSE

You call FCLOSE to release access to the device, some FCONTROL options exercised while the device was open are not reset by 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 — 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 or formfeed.

```
fclose(pfnum,0,0);
fs'error'on'ccl(pfnum,9);
pfnum:=0;  <<** I do this for error handling purposes **>>
```

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. You should avoid linefeeds that follow carriage returns because garbage characters will be echoed to the terminal. (The inbound linefeed collides with the outbound linefeed coming as a result of the carriage return.)

```
ilen:=fread(pfnum,ibuff',precsize);
fs'error'on'ccl(pfnum,2);
if >
  then ; <<** handle eof **>>
```

You may want to trap certain errors returned by FREAD to your program: 22, software time-out; 31, end of line (alternate terminator); and 33, data lost.

ATC — The characters NULL, BS, LF, CR, DC1, DC3, CAN (Control-X), EM (Control-Y), ESC, and DEL are stripped from the input stream for both session and programmatic devices.

ADCC — The characters BS, LF, CR, CAN (Control-X), and EM (Control-Y) are stripped from the input stream for session devices. The characters BS, CR, and CAN (Control-X) are stripped from the input stream for programmatic devices.

The default parity cases are handled quite differently between the ATC and ADCC, you should exercise extreme caution when dealing with parity on the ADCC.

ATC — If the ATC is in the odd/out, no check/in mode all incoming characters have their parity bits set to zero. The same is true for even/out, no check/in mode.

ADCC — If the ADCC is in pass thru/out/in mode all incoming characters retain their parity bits, they are not set to zero. All special characters must have a zero parity bit to be recognized. If the ADCC is in even/out, check even/in mode the incoming characters must have proper even parity and their parity bits are set to zero. The second time you open this terminal the ADCC has switched to pass thru mode and all incoming characters retain their parity bits!!!

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 a device reacts negatively to the DC1, use termtype 18 which suppresses the DC1 on terminal reads. The device must not send data to the HP3000 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 HP3000 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.

```
getprivmode;  <<** necessary for nobuf, no-wait i/o **>>
pifnum:=fopen(pifname,%204,%4404,precsize,pidevice);
if <
  then begin
    getusermode;
    file'error(pifnum,1)
  end;
getusermode;
pofnum:=fopen(pofname,%604,%404,precsize,podevice);
fs'error'on'ccl(pofnum,1);
...
ilen:=fread(pifnum,ibuff',precsize);
fs'error'on'ccl(pifnum,2);
fwrite(pofnum,obuff',-olen,%cctl);
fs'error'on'ccne(pofnum,3);
iowait(pifnum,ibuff',ilen);
fs'error'on'ccne(pifnum,22);
```

When you attach your device to the two ports, connect pin 2, Transmit Data of the terminal to pin 2 of the read port, connect pin 3, Receive Data of the terminal to pin 3 of the write port, and pin 7, Signal Ground of the terminal to pin 7 of both ports. (This two port scheme was first introduced to me by Jack Armstrong and Martin Gorfinkel of LARC.)

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. Some FCONTROL calls shown below affect how FWRITE works. Control returns to your program from FWRITE as soon as the data is loaded into the terminal buffers, it does not wait until all data has been output to the device.

ATC — Carriage control %61 is output as carriage return, formfeed (termtype 10).

ADCC — Carriage control %61 is output as formfeed (termtype 10).

The default parity cases are handled quite differently between the ATC and ADCC, you should exercise extreme caution when dealing with parity on the ADCC.

ATC — If the ATC is in odd/out mode all outgoing characters are given odd parity, even parity is generated when the mode is even/out. Simple.

ADCC — If the ADCC is in pass thru/out mode all outgoing characters retain their parity bits as passed to FWRITE. If the ADCC is in even/out mode all outgoing characters are given even parity. The second time you open this terminal the ADCC has switched to pass thru/out and all outgoing characters retain their parity bits!!!

```
fwrite(pfnum,obuff',-olen,%cctl);
fs'error'on'ccne(pfnum,3);
<<** eof here is probably an error, I mean what is going on? **>>
```

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. 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.

```
fsetmode(pfnum,4);
fs'error'on'ccl(pfnum,14);
```

FCONTROL

FCONTROL is the workhorse intrinsic for managing a programmatic device on the HP3000. Each use of FCONTROL which 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 to your program. 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 the timer is stopped when the DC2 is received from the terminal, a new timer is then started which is independent of the timer set by this FCONTROL call. See the section below on enabling/disabling user block mode transfers.

```
fcontrol'parm:=30;  <<** 30 second time-out **>>
fcontrol(pfnum,4,fcontrol'parm);
fs'error'on'ccl(pfnum,413);
ilen:=fread(pfnum,ibuff',precsize);
if <
  then begin
    fcheck(pfnum,errorcode);
    if errorcode <> 22
      then file'error(pfnum,errorcode*100+2);  <<** something else **>>
    <<** handle time-out **>>
  end;
```

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 — Split speeds are not allowed, FCONTROL 10 and 11 set 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 HP3000 do not expect or desire echoing of the characters they transmit. This option along with FSETMODE 4 completely turns off input echoing. (Control-X is handled separately.) Echoing is not restored when a file is closed so you should always put echo back the way it was found.

```
fcontrol(pfnum,13,prev'echo);
fs'error'on'ccl(pfnum,1313);
...
<<** turn echo back on if it was previously on **>>
if prev'echo = 0
  then begin
    fcontrol(pfnum,12,prev'echo);
    fs'error'on'ccl(pfnum,1213)
  end;
```

FCONTROL – 14, 15 – Disable/enable system break

The break key is typically disabled when terrible things will happen if 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 the 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 prodedure 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.

ATC — In programmatic mode Control-Y's are always stripped from the input.

ADCC — In programmatic mode Control-Y is not stripped from the input if subsystem break is enabled.

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 — 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 timeout, 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. Condition code > implies that the read exceeded 655.35 seconds.

```
fcontrol(pfnum,21,fcontrol'parm);
fs'error'on'ccl(pfnum,2113);
ilen:=fread(pfnum,ibuff',precsize);
fs'error'on'ccne(pfnum,2);
fcontrol(pfnum,22,fcontrol'parm);
fs'error'on'ccl(pfnum,2213);
```

FCONTROL – 23, 24 – Disable/enable parity checking

This option enables parity checking on input for the parity sense specified by FCONTROL 36. Parity checking is overridden by binary transfers (FCONTROL 27) or unedited mode (FCONTROL 41).

ATC — This option affects input parity checking only, output parity generation is controlled by FCONTROL 36.

ADCC — 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.

ATC — Backspace, linefeed, carriage return, DC1, DC3, Control-X, Control-Y, NULL, 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 HP3000 drops the DC2 and sends a DC1 back to the terminal, it 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 — Backspace, linefeed, carriage return,

Control-X, Control-Y, and NULL are not allowed as terminators. The manual claims that DC1, DC3, ESC, and DEL are not allowed as terminators, but they work. DC2 is allowed as a terminator but produces bizarre results unless unedited terminal mode (FCONTROL 41) is also enabled in which case the DC2 is recognized as a terminator in any position.

If a line terminates with an alternate terminator, it will be included in the input buffer and length and an error will be indicated for the read. You must call FCHECK to determine that the read terminated with the alternate character.

```
fcontrol'parm:=[8/0,8/"."];  <<** period is alternate terminator **>>
fcontrol(pfnum,25,fcontrol'parm);
fs'error'on'ccl(pfnum,2513);
...
ilen:=fread(pfnum,ibuff',precsize);
if <
  then begin
    fcheck(pfnum,errorcode);
    if errorcode <> 31
      then file'error(pfnum,errorcode*100+2);  <<** something else **>>
    <<** handle alternate terminator **>>
  end;
```

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 inputting all characters requested, a carriage return or alternate terminator will not terminate the read. No cr/lf is echoed to the terminal at the end of 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. See the note under FCONTROL 25 about DC2 as the first input character on a line. 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 3000 is for the HP3000 to send a DC1 to the terminal indicating it 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 HP3000 responds with another DC1 when it is really ready to take the data, and the terminal sends the data. All of this is transparent to your program which just issues a big read. If your 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 around before the data shows up.

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 the block terminator from the terminal gets lost. This situation is handled by the driver for you, the portion of the read after the second DC1 is sent to the terminal is timed for (#chars in read/#chars per sec)+30 seconds. If the terminator is lost and the read times out, the read will fail and FCHECK will return error 27.

```
fcontrol(pfnum,29,fcontrol'parm);
fs'error'on'ccl(pfnum,2913);
...
ilen:=fread(pfnum,ibuff',-1);
fs'error'on'ccne(pfnum,2);
<<** meddle/muddle **>>
ilen:=fread(pfnum,ibuff',precsize);
fs'error'on'ccne(pfnum,2);
```

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 c ESC H 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. The portion of the read following the second DC1 is timed as described under FCONTROL 28, 29.

FCONTROL – 34, 35 – Disable/enable line deletion echo suppression

This option suppresses the !!!cr/lf echo whenever a Control-X is received from the terminal, the Control-X still deletes all data in the input buffer.

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, no parity, all 8 bits of the data are passed thru; 1, no parity, the parity bit is always set to

one; 2, even/odd, even parity is generated if the original parity bit of the data was a zero, otherwise odd parity is generated; and 3, odd parity, odd parity is generated on all characters.

ATC — FCONTROL 36 sets the parity sense and enables output parity generation. FCONTROL 24 must be called to enable parity checking on input. An undocumented effect of this FCONTROL call is that the previous parity setting is returned in the controlcode parameter wiping out its original value!

ADCC — 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. An undocumented effect of this option is that the previous parity setting is returned in the controlcode parameter wiping out its original value!

Parity is not reset to the default case 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.

ATC — The following results were obtained when parity generation was enabled on output. All options performed as described in the manual.

ADCC — The following results were obtained when parity generation was enabled on output. Option 0, parity pass thru, resulted in even parity on all characters. Option 1, parity forced to one, resulted in odd parity on all characters. Option 2, even/odd parity, resulted in even parity on all characters regardless of the original parity bits of the characters. Option 3, odd parity, resulted in odd parity on all characters. Only option 3 performed as expected.

ATC, ADCC — The following results were recorded when parity checking was enabled on input. Option 0, parity pass thru, resulted in parity errors on all input except that with even parity. Option 1, parity forced to one, resulted in parity errors on all input except that with odd parity. Option 2, even/odd parity, resulted in parity errors on all input except that with even parity. Option 3, odd parity, resulted in parity errors on all input except that with odd parity. Options 2 and 3 performed as expected, options 0 and 1 did not. In all cases, parity bits are always set to zero before the data is passed to your program buffer.

HP has told me that the following is the parity story as of the C-delta version of MPE-IV.

ATC — Options 0 and 1 will not check parity on input, everything else as described above.

ADCC — Option 0 and 1 will be parity pass thru, everything else as described above.

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 even though the manual claims that you do. This option is still useful because it allows you to 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 in the i/o tables. Using this option allows use of a file equation redirecting the program to another device that might not be properly configured.

```
fcontrol'parm:=[11/speed,5/type];
fcontrol(pfnum,37,fcontrol'parm);
fs'error'on'ccl(pfnum,37);
```

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 about the most useful FCONTROL option used to communicate with programmatic devices. It allows almost all control characters to pass through to the HP3000 but does not require 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.

ATC — Unedited terminal mode overrides input parity checking, no checking is performed and all input parity bits are set to zero. Output parity generation is performed normally.

ADCC — Unedited terminal mode processes parity in the same manner as edited mode, see the section on FREAD for an explanation.

Binary transfers enabled overrides 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 on the ATC or ADCC. See the note under FCONTROL 25 about using DC2 as a terminator.

PTAPE

The manual describes PTAPE as the intrinsic to use to read paper tapes. (A fancy data-entry media that is becoming increasingly popular.) It can be used on the HP3000 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 HP3000 the worst strategy is to write some code on the 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 the code, and then the code and device together. I test the device by plugging it into an HP-2645 (or equivalent) terminal, turning on monitor mode, and simulate the HP3000 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 HP3000 to access the device in the manner determined by the first tests. Then plug the HP-2645, not the device, into the HP3000. Now type on the 2645 to simulate the device, continue until your code is debugged. Now you can plug the device into the HP3000 and you have a good (modulo Murphy) chance of actually getting it to work.

REFERENCES

Communications Handbook, Hewlett-Packard Company, April 1981 Part #30000-90105. This manual supersedes the HP Guidebook to Data Communications and the Data Communications Pocket Guide.

Don Van Pernis, "HP3000 Series II Asynchronous Terminal Controller Specifications," Computer Systems Communicator, #15, December 1977, page 2. This explains the terminal subtypes and signal requirements for the ATC.

Charles J. Villa, Jr., "Asynchronous Communications Protocols," Journal of the HP General Systems Users Group, Volume 1, #6, March/April 1978, page 2. Good introductory material.

John Beckett, "Poor Man's Multidrop," Journal of the HP General Systems Users Group, Volume 2, #1, May/June 1978, page 7. How to hook several terminals to the same port.

Tom Harbron, "Lightning, Transients and the RS-232 Interface," Journal of the HP General Systems Users Group, Volume III, #3, Third Quarter 1980, page 14. How to avoid being zapped.

MPE Intrinsics Manual, Hewlett-Packard Company, January 1981 Part #30000-90010. Chapter 5 discusses most of the FCONTROL options that are applicable in terms of the ATC, it is often inaccurate in describing the ADCC.

Everything You Wanted to Know About Interfacing to the HP3000 PART II

John J. Tibbetts
Vice President, Research & Development
The DATALEX Company

INTRODUCTION

The title of the this talk is "Everything You Wanted to Know about Interfacing to the HP3000-Part II." In Part I, Ross Scroggs described in great detail characteristics of the internals of the asynchronous communications protocol, especially for the benefit of those who would wish to tie foreign devices onto an HP3000 through the asynchronous port. This talk — the second part — is intended to take that discussion into a specific direction and discuss how, specifically, to connect intelligent devices, in particular microcomputers, to the asynchronous communications protocol of the HP3000. Note immediately that we are restricting our discussion of microcomputer communication to the asynchronous communications protocol. The reason for this is simply that most microcomputers are easily configurable to communicate asynchronously. Few microcomputer hardware and software packages have been assembled so far which will use bisynchronous communications protocol. Consequently, the reality of the current state of microcomputers suggests that asynchronous communications protocol will be the standard way of hooking up your microcomputer to the HP3000.

This talk will have two major parts. In the first part, we will discuss what is at issue in terms of features and capabilities in a remote communications program. We will discuss in some detail what our standard approaches to handling such capabilities as terminal emulation, sending and receiving files, simultaneously printing to a local printer, and control of the communications protocol from either the local end — that is, the microcomputer end — or from the remote computer end. In this talk we will refer to the local side as being the microcomputer and the remote side as being the remote or the host computer.

In the second part of the talk we will describe how you can actually get such a program running on your own machine. The choices are twofold: either buy one or write one. By using the criteria we have established in part one of the talk, we will try to outline some of the considerations of doing either of these.

One final note before we begin is that the remote communications capabilities tend to be a very hardware- and software-specific part of microcomputing. Whereas one can usually take a CPM program written in BASIC, for instance, and run it on most or all CPM implementations, one cannot expect to do the same with remote software. Remote software usually has to talk to pieces of your microcomputer which the operating system tends not to know anything about. During the course of the talk we will periodically make reference to a specific capability that is required for the remote program and in the published paper we will annotate them as a capability bullet that you will need to either have supplied to you or you will have to implement on your microcomputer to get this particular capability to be implemented for your remote program.

TERMINAL EMULATION

The first and perhaps the easiest capability to implement on your microcomputer is the emulation of a simple terminal.

- Capability — handling the remote port. Any of the operations we will be discussing for remote communications program presume that your microcomputer has a separate usable asynchronous communications port. Your program should be able to perform the following operations on that port:
  - read a character
  - write a character
  - test to see if a character is ready to be read.

  This last capability is the one that is usually missing in the standard microcomputing operating environments. In particular, CPM implements the read and write character routines as the reader and punch devices, respectively, but do not have a standard driver entry for testing the status of the remote port. Usually, you have to specifically write this capability for your own hardware if it hasn't been provided to you by someone else.

The standard procedure for implementing a terminal emulator is to write a polling loop. In the polling loop a very tight program loop tests to see if a character has

been entered, either at the remote port or at the keyboard. If the character has been entered on either one, the character is then read and written to its opposite port. Thus, a character entered at the keyboard of the microcomputer, when sensed, would be written to the remote port and vice versa. It is important to make the polling loop as quick as possible. No code should be included in that loop unless it is absolutely necessary. Especially if you are writing in a high level language and especially if that language is interpretive, such as BASIC, you may have speed problems when trying to emulate a terminal at higher baud rates, say over 2400 baud. When writing in assembly language this is usually less important and you will find that you can support virtually any standard baud rate.

When emulating a half duplex terminal, any character entered at the keyboard should be immediately written back to the screen, thus providing the local echo. If terminal emulation requirements stopped here, a terminal emulator would be a very easy piece of software to write. Unfortunately, there are usually a few special problems with the terminal emulator.

The first problem is handling the break key. Many timesharing systems do not require break keys, but as any member of the HP3000 audience knows, the break key is a very crucial part of terminal handling on the HP3000. Unfortunately, the break key is not a character in the way any other keystroke on the terminal is. When depressed, the break key actually changes the electrical state of the transmit pin.

- Capability — sending a break. Most microcomputer communication ports have a mechanism by which the output port can be put into a break state. It almost always requires assembly language programming to implement a break key function. The actual reasoning behind break key handling goes beyond the scope of this talk. Suffice it to say that the preferred technique of break key transmission is, when the break key of a terminal is sensed, to put the output port into the break state until either 200 milliseconds have elapsed or a character comes in on the remote port.

Thus, to properly handle the break key our polling loop now needs to be expanded to test to see if the value entered from the keyboard is the break signal. When the break signal is sensed, the polling loop should then, instead of sending that character, invoke the send break routine to send a break.

The second capability which makes the terminal emulator more difficult relates to simultaneous printing of the terminal interactions on a printer which is hooked up to the microcomputer. The whole issue of printers, and especially printers that might hold up communications flow, is dealt with in a subsequent section.

SENDING AND RECEIVING FILES

Probably the main, useful work we would like for a communications program is to send files from our microcomputer to our HP3000 and receive files from the HP3000 down to the microcomputer. At first glance this may seem to be a rather simple operation. To send the file we should simply read the file from the local storage medium on the microcomputer and write it out the remote port. To receive a file we should simply read from the remote port and write to the diskette. If it were only so simple . . . There are three issues which will significantly complicate the issue of sending and receiving files. They are:

1. The vast majority of computers need time for themselves. What I mean by this statement is that at various times in the life of a computer it needs time to handle data it has been sending or receiving. On an HP3000, if you should try to type characters into it before it has put a prompt character up, you know that you will lose those characters. On a microcomputer, if you try to enter characters into most microcomputers while it is reading and writing a diskette file, for example, those characters will be lost. These phenomena reflect the fact that most computers are not designed to be able to handle communications of their terminal or remote ports at any time they are activated. A newer line of more commercially oriented microcomputers are beginning now to feature interrupt systems that do have full functioning typeahead systems which greatly ameliorate these problems. However, these microcomputers are definitely in the minority. Thus, our communications program needs to somehow be able to allow each computer to have time for itself when it needs it.

2. Communications lines tend to be rather noisy, especially if we are using the telephone system to transmit our data. Since file integrity is usually important, we need to come up with some kind of error checking protocol which can detect errors in the transmission of the data being sent or received. Interestingly enough, most of the programs running now on microcomputers for sending and receiving data do not handle error detection. The reason is that so far most microcomputer users who are using communications programs are doing so to make use of timesharing networks such as The Source for sending and receiving programs. As more, real, data processing functions, which might relate to shared databases or distributed data entry, are being built, clear data transmission protocols will become very important.

3. Most systems have some kind of difficulty with binary transmissions. This may not be a problem in applications in which only textual data needs to be sent. As time goes on, one finds the need to send binary data, for instance, to distribute object code of programs through the communications program. Thus it becomes desirable to be able to send binary files.

This is a summary of the problems — now let's take a look at some of the possible solutions.

MESSAGE HANDLING

Message handling is the general title by which we refer to the problem of the traffic control of the data being passed back and forth between the micro and the HP3000. The message handling protocol determines when data can safely be sent or received so that we never go faster than either of the machines can accommodate. The very first thing that becomes apparent after some investigation and experimentation is that the send and receive case are quite different from one another with *this* pair of computers. This is unusual when compared to communications software usually existing between microcomputer and microcomputer. In that case, the communications message handling is usually symmetric; that is, whatever convention is used to control data flow on the send side is also used symmetrically in the other direction to control it on the receive side. We have to do extra work on the HP3000 since none of it asynchronous communications protocol was designed for access by an intelligent terminal, and it ends up having some asymmetric properties which we have to deal with.

First, let's consider the case of sending data from a microcomputer to an HP3000. The first fact one must always be aware of when trying to send data to an HP3000 through its asynchronous communication port is that it can only read data from a device when a read is up; that is, when a read has been issued from a program. If you try to type ahead on an HP3000, the data is lost. Fortunately, in the HP3000 communications software a character is always sent whenever a read is put up. That character is the Control-Q or the DC1 character. Thus, any device trying to send data to the HP3000 can simply wait until it sees a DC1 and then send its record of data terminated by a carriage return. This type of *data interlocking* is the preferred method of sending data to an HP3000. For instance, this is the mechanism that LINK-125 uses in its protocol. It simply invokes FCOPY, and when FCOPY puts up its first read, it hands a record of data to it, terminates it with a carriage return, and proceeds with file transmission in that fashion.

But this is not always good enough. Consider this case: a message has been transfered to the HP3000, a carriage return sent following it, the HP3000 has issued another read, has sent the DC1 back along the phone line and suddenly there is a noise burst on the phone line. The DC1 coming back to the microcomputer is lost. The microcomputer is waiting there to transmit its next record of data with the DC1 and then deadlocks because it never sees the DC1. This type of deadlocking is characteristic of trying to make too much out of a simple interlocking protocol. What we really find as more desirable is to write a communications program on the HP3000 which talks to the program on the microcomputer. This will allow the microcomputer program and the HP3000 program to issue reads with timeouts which would require that after a certain amount of time we give up on a particular read because of lost protocol characters or a dropped line. In the particular case of the missing DC1 — and this is only one of the pathological conditions that can arise — the microcomputer program can simply, after a certain amount of time, send a message up the line which says something like, "Hey, are you still there?" to which the HP3000 program will response, "Yes, I am still here and here is another DC1" or may not respond at all if the machine has crashed or the line has gone down. This concept of using programs on both sides is really what differentiates very simple dumping of files up and down the line from more sophisticated communications protocols. I feel that this approach is required for any serious use of communications, especially with any bulk of data transmission which we would like to move reliably back and forth. Using this "program-to-program" approach, we can also perform some other more sophisticated error checking which we will get into shortly.

As we leave the send case, note this important fact. Make sure that after a record has been sent to the HP3000 with its carriage return the very next piece of work the microcomputer does is to turn around and wait for the DC1 before it does anything else. One might be tempted to put up the next read to the diskette to pull the next record off while waiting for the DC1. If your microcomputer has the appropriate communications typeahead software on its remote port, you might be able to get away with this. However, in general the micrcomputer needs to wait for that interlock character to come back before it tries to do any other useful work. Otherwise, you will start missing DC1s and your program will get hung up.

RECEIVING FILES

Just as we have done with the send case, let's examine the most trivial method of receiving files which would not rely on a program being run on the HP3000 side. The basic fact of life when receiving files is that the remote computer — the HP3000 — will be instructed to start sending down a file; perhaps we use FCOPY or the editor to start sending a file to us. The microcomputer is going to periodically need to write out the buffer it is accumulating to the disk drive. When it does this there will usually be a second in which it can't receive any data. The first approach is to just receive small files, in which case the microcomputer never writes out its data until it has collected the full file in memory. This of course limits the size of the file you can receive to the amount of available memory on the microcomputer, usually somewhere between 10,000 and 40,000 bytes. Obviously, this is an unsatisfactory method of receiving files unless your application is very limited. The next idea that comes to mind is making use of the X-on/X-off characteristics of the HP3000 to control this flow. As a human user, sometimes when a listing is coming out too quickly onto the CRT, we stop the flow by typing the

X-off key which is a Control-S and most of the time the HP3000 stops its transmission flow until you have done what you wanted and then you hit a Control-Q and the scrolling of the data output continues. Maybe we could have the microcomputer perform this function for us as a simple interlocking method.

The answer is that "Yes, we can," however, it is not the preferred method of receiving files. The reason for this is that, surprisingly, Control X-on/X-off protocol seems to have some holes in it on the HP3000 side. Someone told me that after some extensive testing they found that one out of five X-off characters seems to drop into a hole when sent to the HP3000. I have absolutely no way of verifying this other than to tell you it has happened a number of times to me. This doesn't make the use of this mechanism impossible, it simply complicates it somewhat.

Using this technique then, what you need to do is:

1. Build a large receive buffer.
2. Start receiving data until the buffer gets to 80% or 90% full.
3. Send an X-off character to the HP3000 but keep receiving the characters onto your microcomputer.
4. After some predetermined timeout time — perhaps a second of no characters coming in — assume it has finally absorbed the X-off character, and then you can proceed with your disk writes of the buffer.

But, if 4 or 5 characters have passed without stopping, send the X-off character again. Repeat these steps until the data transmission actually stops.

Just like the send file case, I recommend the use of a program on both sides. Using this technique we will simulate the kind of data interlocking protocol that the HP3000 uses. That is, every time the microcomputer is ready to issue a read to the remote HP3000 it will issue a character, perhaps for symmetry's sake a Control-Q, or any character of your choice. When that character is received by the program on the far side, that program will then send down the next record of data to the microcomputer followed by some standard termination character. After the message has been received, the microcomputer can set to work writing that message to the disk or doing whatever other housekeeping it would like to do. It then issues the next interlock character, and proceeds. This protocol also allows for the kind of timeout mechanisms that I described in the send case so that you can recover from lost transmission and especially lost protocol transmission. It will also easily accommodate the kind of error checking we will be talking about in the next section.

As always, there is a complication and a warning. Even in the case we have just described, we have not really built a symmetric communications protocol to the HP3000. The interlock character itself, which is going to be sent to the HP3000, has to be read by an HP3000 read. Of course, that HP3000 read will have a DCl coming right before it and any attempt to write the protocol character up the line before the HP3000 is able to read it results in a lost protocol character. Thus, some real world experimentation is usually needed in which some delay is required after the record has been received from the HP3000 so that it will have had time to finish writing the record and then put up the read which will read the next character interlock. It's for reasons like these, incidentally, that interfacing microcomputers to the HP3000 has not always been the simplest and the least frustrating of tasks.

The other item of note is that on reading characters into the microcomputer it is usually wise to strip out occurrences of the protocol characters that have accumulated in the asynchronous communication chip. These characters would be the line feed character which the HP3000 will usually tag onto the end of the carriage return unless you turn that off, and also the DCl character itself. Although these characters will be flying around during the transmision of the data, you don't want to include them into the data stream itself. They should be filtered out of the actual data flow.

ERROR HANDLING

Now that we have described the actual methods by which data can be sent and received, let's go on to the second defined problem in our data communications task — error handling.

The fact is that there are very few asynchronous communications protocols which go to this level, and I find this fact to be extremely regrettable. No serious large-scale interface of microcomputers to any kind of data processing network can be accomplished without real error checking. However, once we have built the proper send and receive frameworks with the right kinds of interlocking and assuming there is some intelligence and flexibility on both ends, it becomes rather easy to add the error handling phase. What are some of the usual techniques for adding error handling to the send and receive cases that we've described?

The standard mechanism, of course, is to add to each message sent or received some kind of check character or checksum which is used to check out the validity of the data. The simplest form of a checksum is an addition of the various character values of the message. For instance, if one record of data I am sending to the HP3000 is 40 characters long, the microcomputer can run through those 40 characters, add up the ASCII value of the 40 characters, and then produce a new character for the string and tag it onto the beginning or the end of that string. The HP3000 on the other end, when it has received the data, will go through the very same operation except that this time it will strip the character off and compare it with its own calculation of that string and see if they match.

There are some problems with the simple add-em-up

checksum and there are many other sophisticated algorithms around — I can refer you to literally any book on communications for a description of CRC algorithms. The problem with CRC algorithm is that it's usually a fairly difficult algorithm to execute quickly enough on a microcomputer unless you are programming in an assembler language. The algorithm I have found to be very simple but very effective is an algorithm which adds and shifts the bits as the characters roll in. In this algorithm each character is added on to the checksum and then the checksum is multiplied by 2 which shifts all the bits to the higher order by one bit. Then it receives the next character and repeats the process. This final 16-bit quantity is then tagged onto the message.

You have to remember not to freely insert binary integers into the communications stream. Some adjustment of the value must be done when we are sending it to the HP3000 to make certain we are not sending a character it will have difficulty receiving.

Once a message has been sent to the other side with a checksum on it, the other side has the opportunity to examine that message and respond. The typical response is for the receiver to send back some predetermined character message which says either: the data was received successfully and you should proceed to the next block; or, alternatively, the data was not received correctly so retransmit the block just transmitted. I usually include a third state in this message traffic which indicates that something terrible has happened on one end or the other and to abort the entire transmission process altogether. You can include in this function the ability for the user to hit some kind of escape key and abort the communicatons traffic.

One other item I have found to increase reliability is to add sequence numbers on each of the messages sent or received. This would ensure that in some pathological case we don't actually get the blocks out of order; that is, in a case where an entire block has dropped out of the communications traffic. Although this is fairly rare, there are actually certain conditions which can cause something like that to happen. A sequence number which is checked on both sides for each block transmitted can protect against this possibility.

BINARY TRANSFERS

We have mentioned previously that it is generally unreliable to transmit 8-bit binary characters from a microcomputer up to an HP3000. What are the possible ways around this problem? The standard way is to simply convert the 8-bit binary traffic into hexadecimal strings, that is convert a binary character 255 into the ASCII string FF, etc. Of course, you would probably immediately see this means that there is a 50% reduction in communications efficiency. This technique is usually simple to perform and it is useful when the binary traffic is somewhat limited. A technique that I prefer is to translate seven 8-bit bytes into eight 7-bit bytes. This is quickly accomplished by gathering the 8th level bit of the 7 bytes input and building another byte and tagging it onto the back end of each 7-byte block. This effectively chops the 8th level off the communications stream at transmission time and is then reassembled on the far side. If this technique is used on the entire message, including checksums, sequence numbers or any kind of message identifier on the block, the whole communications interface becomes considerably simpler.

USING PRINTERS

It is often desirable in a communications protocol to log the data to the printer. For instance, on receiving a file to a microcomputer you may want to get a listing of it. Alternatively, you may wish, during terminal emulation, to get a copy of that session onto a hardcopy printer.

Like everything else mentioned in this talk, there are hidden catches. It seems simple enough to be able to put in a switch in the software — for instance, in the polling loop of the terminal emulator — that when a character has been sent or received, it should be sent to the printer port. However, many printers don't print at the communications speed. We will therefore distinguish between a fast printer and a slow printer. In this context, fast and slow do not have any absolute meaning to them. Fast means that the printer operates faster than the current communications context, and slow means that the printer operates slower than the current communications context. For example, in a 300-baud environment most printers (for example, an Epson matrix printer or a TI-810) will be fast printers. However, at 1200 baud most of the inexpensive matrix printers are slow printers, that is, they cannot keep up with the 1200-baud stream. Surprisingly, even printers such as the TI-810 which are rated at from 120 to 150 characters per second often cannot keep up with the 1200-baud flow of data. Therefore, the determination of whether a printer is a fast or slow printer can only be done by running a series of tests.

As you might now be able to suspect, there is very little difficulty with a true, fast printer in our communications program. Any characer we wish to print we simply output to the printer port. However, on a slow printer we have to do more resource balancing in that there is now another resource in the communications environment which needs time of its own. Adding a slow printer to a communications program can easily double the complexity of the communications environment.

At this point let me summarize a few of the major elements of printer integration:

- If the communications program has been implemented, as I have been suggesting, with a pair of programs on either end which have an interlocking mechanism, the simplest approach of integrating a slow

printer is to print out the block of data during the time that the program is performing activities such as writing to the diskette or reading from the diskette. That is, after the message has been sent or received and before the interlock causes the pair of programs to proceed, the buffer of data sent or received can be put to the printer. An unfortuante side effect of this approach is that the printer is only printing between records. This does not take advantage of the fact that there may be sufficient time during the actual communications transmission to have the printer doing some useful work.

- Improvement on this scheme requires a new capability:

  Capability — Printer Ready — The printer ready capability says that our communications software can sense when the printer is available; that is, when a character can be written to the printer in such a way that the printer buffer will absorb the character instantly.

With a printer-ready capability in our software, we can build a more sophisticated operating environment in which we have, in effect, a small spooler being operated. That is, any data which has been successfully sent or received and is ready to print can be added to a print buffer. This buffer is metered out to the printer when the printer is ready. It is important that the remote communications facility always have top priority. The other mechanism that needs to be in effect in this type of environment is that as the printer buffer gets close to being full, a flag will go up which will hold the interlock the next time around until the print buffer has been totally cleared. Although this mechanism sounds somewhat obtuse, it actually provides a very effective method of integrating a slow printer into a communications environment.

- Integrating a slow printer into the terminal emulator mode can be accomplished by using the X-on/X-off character techniques I described in the receive section. That is, if printing is active, a mechanism will go into effect, during terminal emulation mode, such that the microcomputer dispatches X-on/X-off to control the characters coming from the remote computer into the microcomputer.

BIDIRECTIONAL CONTROL

The last major capability we will discuss in our communications program is the ability for the remote computer to assume control of the communications program. This can be very desirable in applications in which an operator may activate a communications program and get online with an HP3000 and perhaps start a UDC. At that point the UDC might take over all control of the microcomputer through the communications program such that it can request files to be sent and received. The following are a couple of points concerning bidirectional control:

- The basic concept in bidirectional control is that the remote computer can have some escape character which it can send to the microcomputer during terminal emulation that will cause the remote computer to assume command of the microcomputer. Commands can then be dispatched by the remote computer directly to the microcomputer. Be sure that all of the issues previously mentioned about interlocking and protocol are also supported by any direct interaction between the remote computer and the microcomputer.

- It is very useful to be able to have a capability whereby the remote computer can ask for directory listings directly from the microcomputer. This gives the remote computer a list of what files may need to be sent or received.

- You may wish to give the remote program the ability to actually terminate the communications session itself and to remove the user from the terminal emulator mechanism.

IMPLEMENTATIONS

The best thing you can do with communications software is to buy it rather than develop it. Unfortunately, this assumes that someone has developed the type of software running on the type of machine you desire. As we've indicated during the course of this talk, remote communications software tends to be more hardware dependent than almost any other software running on your computer. Not only is it hardware dependent, it is also operating system dependent. Thus, on a single machine — for instance, the Apple which can run the Apple DOS, the PASCAL operating system, and CPM (if the CPM card is added) — each of these three operating systems has a different file system and each has different requirements for its communications program. This means that no one program will solve all of your problems.

Let's consider some of the available implementations. Under CPM, there are a couple of programs fairly well known in the CPM community for doing file-to-file transfers. They are a program called CROSSTALK and a program called COMMX. Both of them are available through the major CPM software distributors. Both of the programs feature a non-protocol mode and a protocol mode. In the non-protocol mode you can easily make the software talk to your HP3000 by setting the DCl character to the interlock character. Unfortunately, on both of these programs the protocol mode which includes the checksumming algorithms is only usable when the program is talking to another CPM program of its own type. Clearly, these programs are written for CPM systems to talk to other CPM systems, not to another computer system. This means that if you do wish to turn one of these systems into a protocol checked operating environment, you need to do a little extra work on it. If you have an HP125 you can acquire LINK-125 which does a good, but not error-checked, link with the HP3000.

None of the programs I have seen feature bidirectional control which would allow, as I have described in

the talk, the remote computer to assume the control of the microcomputer.

If you are running some variant of the UCSD PASCAL or UCSD p-System operating environment, then, with all due modesty, there is no better communications software available than that provided by our own company. It incorporates in a table-driven fashion, ready-to-run for the HP3000, all of the capabilities described in this talk, that is: full error-checked protocol, the ability to support fast and slow printers, full bidirectional control, and blank compression of the data. All of the software for communicating with an HP3000 has been worked out in great detail. Incidentally, we also support protocol-oriented communication for other p-Systems — that is, for p-System to p-System communication — as well as communication with the IBM 370 interactive operating system such as CMS, CSS, or TSO, and DEC-10, -11, and -20 support.

Something new is that the software distributors for the UCSD p-System now have a CPM file compatibility mode which, when available, will mean that we can also use our communications software to send and receive CPM files as well.

CONCLUSION

If there is any theme for a discussion of communications software, it is *"There is More Than Meets the Eye."* As I have repeatedly stressed, the very best way of solving your communication problems is finding someone else who has already solved them and acquire the software from them. This is my very strong recommendation when attempting to establish a remote communications network for your system.

I would also refer to the other talk I am giving at this meeting which encompasses distributed processing applications using microcomputers. It is entitled "Microcomputer-based Transaction Processing with Your HP3000" and it goes into some detail about the state of the art in microcomputer software for distributed processing.

Programming for Device Independence

John Hulme
Applied Cybernetics, Inc.
Los Gatos, California

INTRODUCTION

The purpose of this presentation is to discuss techniques and facilities which:

1. Isolate the programmer from specific hardware considerations
2. Provide for data and device independence
3. Allow the programmer to deal with a logical rather than a physical view of data and devices
4. Allow computer resources to be reconfigured, replaced, rearranged, reorganized, restructured or otherwise optimized either automatically by system utilities or explicitly by a system manager or databse administrator, without the need to rewrite programs.

The evolutionary development of these techniques will be reviewed from a historical perspective, and the specific principles identified will be applied to the problem of producing formatted screen applications which will run on any type of CRT.

WHAT IS A COMPUTER?

As you already know, a computer consists of one or more electronic and/or electromechnical devices, each capable of executing a limited set of explicit commands. For each type of device some means is provided to allow the device to receive electrical impulses indicating the sequence of commands it is to execute. In addition to commands, most of these devices can receive electrical impulses representing bits of information (commonly called data) which the device is to process in some way. Nearly all of these devices also produce electrical impulses as output, which may in turn be received as commands and/or data by other devices in the system.

Nowadays, most devices also have some form of "memory" or storage media where commands or other data can be recorded, either temporarily or semi-permanently, and a means by which that data can later be received in the form of electrical impulses.

The tangible, visible, material components which these devices are physically made up of is generally called computer *hardware*. Any systematic set of instructions describing a useful sequence of commands for the computer to execute can be called computer *software*. As we will see later, software can be further subdivided into *system software*, which is essentially an extension of the capabilities of the hardware, and *application programs*, which instruct the computer how to solve specific problems, handle day-to-day applications, and produce specific results.

Originally it was necessary for a computer operator to directly input the precise sequence of electrical signals by setting a series of switches and turning on the current. This process was repeated over and over until the desired sequence of instructions had been executed.

By comparison with today's methods of operating computers, those earlier methods can truly be called archaic. Yet the progressive advancement of computer systems from that day to this, however spectacular, is nothing more than a step-by-step development of hardware and software building blocks, an evolutionary process occurring almost entirely during the past 25 years.

ENGINEERING AND AUTOMATION

I think we mostly take for granted the tremendous computing power that is at our fingertips today. How many of us, before running a program on the computer, sit down and think about the details of hardware and software that make it all possible? For that matter, who stops to figure out where the electrical power is coming from before turning on a light or using a household appliance? Before driving a car or riding in an airplane, who stops to analyze how it is put together?

Probably none of us do, and that is exactly what the design engineers intended. You see, it is the function of product engineering to build products which people will buy and use, which usually means building products which are easy to use. The fact that we don't have to think about how something works is a measure of how simple it is to use.

Wherever a process can be automated and incorporated into the product, there is that much less that the consumer has to do himself. Instead of cranking the engine of a car, we just turn a key. Instead of walking up 30 flights of stairs, we just push a button in the elevator.

It's not that we are interested in being lazy. We are interested in labor-saving devices because we can no longer afford to waste the time; we have to meet deadlines; we want to be more efficient; we want to cut costs; we want to increase productivity. We also want to reduce the chance for human error. By automating a

complicated process, we produce consistent results, and when those results are thoroughly debugged, error is virtually eliminated. We can rely on those consistent results, which sometimes have to be executed with split second timing and absolute accuracy. Without reliable results there might be significant economic loss or danger to life and limb. Imagine trying to fly modern aircraft without automated procedures.

Automation also facilitates standardization, which allows interchangeability of individual components. This leads to functional specialization of components, which in turn leads to specialization of personnel, with the attendant savings in training and maintenance costs. And because the engineering problem only has to be solved once, with the benefits to be realized every time the device is used, more time can profitably be spent coming up with the optimum design.

BUILDING BLOCKS

In my opinion, the overwhelming advantage of automating a complicated process is that the process can thereafter be treated as a single unit, a "black box," if you will, in constructing solutions to even more complicated processes.

Later, someone could devise a better version of the black box, and as long as the functional parameters remain the same, the component could be integrated into the total system at any time in place of the original without destroying the integrity of any other components.

It is this "building-block" approach which has permitted such remarkable progress in the development of computer hardware and software. As we review the evolution of these hardware/software building blocks, keep in mind that the chronological sequence of these developments undoubtedly varied from vendor to vendor as a function of how each perceived the market demand and how their respective engineering efforts progressed.

ONE STEP AT A TIME

Even before the advent of electronic computers, various mechanical and electro-mechanical devices had been produced, some utilizing punched card input. Besides providing an effective means of input, punched cards and paper tape represent a *rudimentary storage medium*. Incorporating *paper tape and card readers* into early computer systems not only allowed the user to input programs and data more quickly, more easily, and more accurately (compared with flipping switches manually), but on top of that it allowed him to enter the same programs and data time after time with hardly more effort than entering it once.

The next useful development was the *"stored program"* concept. Instead of re-entering the program with each new set of data, the program could be read in once, stored in memory, and used over and over.

This concept is an essential feature of all real computers, but it would have been practically worthless except for one other essential feature of computers known as *internal logic*. We take these two features so much for granted that it's hard to imagine a computer without them. In fact, without internal logic, computers really wouldn't be much good for anything, since they would only be able to execute a program in sequential order beginning with the first instruction and ending with the nth. Internal logic is based on special hardware commands which provide the ability first of all to test for various conditions and secondly to specify which command will be executed next, depending on the results of the test. In modern computer languages, internal logic is manifest in such constructs as IF statements, GO TO statements, FOR loops, and subroutine calls.

But at the stage we are discussing there were no modern programming languages, just the language of electrical signals. These came to be represented as numbers (even letters and other symbols were given a numeric equivalent) and programs consisted of a long list of numbers.

Suppose, for example, that the numbers 17, 11, and 14 represented hardware commands for reading a number, adding another number to it, and storing the result, respectively, and suppose further that variables A through Z were stored in memory locations 1 through 26. Then the program steps to accomplish the statement "give Z a value equal to the sum of X and Y" might be expressed as the following series of numbers, which we will call *machine instructions:*

17, 24, 11, 25, 14, 26

In essence, the programmer was expected to learn the language of the computer.

A slight improvement was realized when someone thought to devise a meaningful mnemonic for each hardware command and to have the programmer write programs using the easier-to-remember mnemonics, as follows:

READ, 24, ADD, 25, STORE, 26

or perhaps even

READ, X, ADD, Y, STORE, Z.

After the programmer had described the logic in this way, any program could be readily converted to the numeric form by a competent secretary. But since the conversion was relatively straightforward, it would be automated, saving the secretary some very boring work. A special computer program was written, known as a *translator*. The mnemonic form, or *source program* as it was known, was submitted as input data to the translator, which substituted for each mnemonic the equivalent hardware command or memory location, thus producing machine instructions, also known as *object code*. Translators required two phases of execution, or two passes, one to process the source program and a second to execute the resulting object code. Once the program functioned properly, of course, it could be

executed repeatedly without the translation phase.

It would have been possible for the hardware engineers to keep designing more and more complicated hardware commands, and to some extent this has been done, either by combining existing circuitry or by designing new circuits to implement some new elemental command. Each new machine produced in this way would thus be more powerful than the last, but it would have been economically prohibitive to continue this type of development for very long and the resulting machines would have been too large to be practical anyway.

Engineers quickly recognized that instead of creating a more powerful command by combining the circuitry of existing commands, the equivalent result could be achieved by combining the appropriate collection of commands in a miniature program. This mini-program could then be repeated as needed within an application program in place of the more complex command. Or better yet, it could be kept at a fixed location in memory and be accessed as a subroutine just the same as if it were actually a part of each program.

Another approach was to use an *interpreter,* a special purpose computer program similar to a translator. The interpreter would accept a source program in much the same way as the translator did, but instead of converting the whole thing to an object program, it would cause each hardware command to be executed as soon as it had been decoded.

Besides requiring only one pass, interpreters had the added advantage of only having to decode the commands that were actually used, though this might also be a disadvantage, since a command used more than once would also have to be decoded more than once.

The chief benefit of an interpreter lay in its ability to accept mnemonics for commands more complex than those actually available in the hardware, and to simulate the execution of those complex commands through the use of subroutines. In this way, new commands could be implemented without any hardware modifications merely by including the appropriate subroutines in the interpreter. This step marked the beginning of system software.

In addition, source programs for nearly any computer could be interpreted on nearly any other computer, as long as someone had taken the time to write the necessary interpreter. Interpreters could even be written for fictional computers or computers that had been designed but not yet manufactured. This technique, though generally regarded as very inefficient, provided the first means of making a program transportable from one computer to another incompatible computer.

It is possible, of course, to apply this technique to translators as well, allowing a given mnemonic to represent a whole series of commands or a subroutine call rather than a single hardware instruction. Such mnemonics, sometimes called macros, gave users the impression that the hardware contained a much broader repertoire of commands than was actually the case.

Implementing a new feature in software is theoretically equivalent to implementing the same function in hardware. The choice is strictly an economic one and as conditions change so might the choices. One factor is the universality or frequency with which the feature is likely to be used. Putting it in hardware generally provides more efficient execution, but putting it in the software is considerably easier and provides much greater flexibility.

The practice of restricting hardware implementation to the bare essentials also facilitated *hardware standardization* and compatibility, which was crucial to the commercial user who wanted to minimize the impact on all his programs if he should find it necessary to convert to a machine with greater capacity. Beginning with the IBM 360 series in 1964 *"families" of compatible hardware* emerged, including the RCA Spectra 70 series, NCR Century series, and Honeywell 200 series, among others.

Each family of machines had its own *operating system,* software monitor, or executive system overseeing the operation of every other program running on the machine. In some systems, concurrent users were allowed, utilizing such techniques as memory partitioning, time-sharing, multi-threading, and memory-swapping. Some form of *job control language* was devised for each operating system to allow the person submitting the jobs to communicate with the monitor about the jobs to be executed.

Introducing families of hardware did not solve the problem of compatibility between one vendor and the next, however, a problem which could only be solved by developing programming languages which were truly independent of any particular piece of hardware.

Since the inventors of these so-called *higher-level languages* were not bound by any hardware constraints, an effort was made to make the languages as natural as possible. FORTRAN imitate the language of mathematical formulas, while ALGOL claimed to be the ideal language for describing algorithmic logic; COBOL provided an English-like syntax, and so on.

Instead of having to learn the computer's language, a programmer could now deal with computers that understood his language. Actually, it was not the hardware which could understand his language, but a more sophisticated type of translator-interpreter known as a *compiler.*

To the degree that a particular language enjoyed enough popular support to convince multiple vendors to implement it, programs written in that language could be transported among those machines for which the corresponding compiler was available.

The term compiler may have been coined to indicate that program units were collected from various sources besides the source program itself, and were compiled

into a single functioning module. Subroutines to perform a complex calculation such as a square root, for example, might be inserted by the compiler whenever one or more square root operations had been specified in the body of the source program.

Embedding subroutines in the object code was not the only solution, however. It became more and more common to have the generated object programs merely "CALL" on subroutines which were external to the object program, having been pre-compiled and stored in vendor-supplied *"subroutine libraries."* This concept was later extended to allow users a means of placing their own separately-compiled modules in the library and accessing them wherever needed in a program.

I should mention that an important objective of any higher level language should be to enable a user to describe the problem he is solving as clearly and concisely as possible. Although the emphais is ostensibly on making the program easy to write, being able to understand the program once it has been written may be an even greater benefit, particularly when program maintenance is likely to be performed by someone other than the original author.

It is well-known that program maintenance occupies a great deal of the available time in the typical data processing shop. Some studies estimate the figure at over 50% and increasing. In order to be responsive to changing user requirements, it is essential to develop methods which facilitate rapid and even frequent program changes without jeopardizing the integrity of the system, and without tying up the whole DP staff.

To avoid having to re-debug the logic every time a change is made, it is often possible to use *data-driven or table-driven programming techniques.* The portion of the program which is likely to change, and which does not really affect the overall procedural logic of the program, is built into tables or special data files. These are accessed by the procedural code to determine the effective instructions to execute.

The most common example in the United States, and perhaps in other countries as well, is probably the table of income tax rates, which changes by law now at least once a year. The algorithm to compute the taxes changes very rarely, if at all, so it does not have to be debugged each time the tables change. In simple cases like this, non-programmer clerks might safely be permitted to revise the table entries.

In more sophisticated applications, tables of data called *logic tables* may more directly determine the logic flow within a program. The program becomes a kind of interpreter, and elements in the logic table may be regarded as instructions in some esoteric machine language. Such programs are generally more difficult to thoroughly debug, but once debugged provide solutions to a broad class of problems without ever having to revise the procedural portion of the program.

Sometimes, logic-controlling information is neither compiled into the program nor stored in tables, but is provided to the program when it is first initiated or even during the course of execution, in the form of *run-time parameters* or *user responses.* The program has to be pre-programmed to handle every valid parameter, of course, and to gracefully reject the invalid ones, but this method is useful for cutting down the number of separate programs that have to be written, debugged, and maintained. For example, why write eight slightly different inventory print programs, if a single program could handle eight separate formats through the use of run-time options?

Incidentally, program recompilations need not always cause alarm. Through the proper use of *COPY code,* programs can be modified, recompiled, and produce the new results without the original source program ever having to be revised. This is made possible by a facility which allows the source program to contain references to named program elements stored in a COPY library instead of having those elements actually duplicated within the program. A COPY statement is in effect a kind of macro which the compiler expands at the time it reads in the source program.

For example, if a record description or a table of values appears in one program, it is likely to appear in other programs as well. It is faster, easier, safer, and more concise to say "COPY RECORD-A." or "COPY TABLEXYZ." than to re-enter the same information again and again. And if for some reason the record layout or table of values should have to be changed, merely change it in the COPY library, not in every program.

By changing the contents of a COPY member in this way and subsequently recompiling selected programs in which the member is referenced, those programs can be updated without any need to modify the source. If procedure code is involved, the new COPY code only need be debugged and retested once rather than revalidating all the individual programs.

Where blocks of procedural code appearing in many programs can be isolated and separately compiled, however, this would probably be better than using COPY code. For one thing, the *separate modules* would not have to be recompiled every time the procedural code was revised.

BITE-SIZE PIECES

Breaking a complex problem into manageable independent pieces and dealing with them as separate problems is a valuable strategy in any problem-solving situation. Such a strategy has added benefits in a programming environment:

1. Smaller modules are typically easier to understand, debug, and optimize.
2. Smaller modules are usually easier to rewrite or replace if necessary.
3. Independent functions which are useful to one application are often useful to another application;

using an existing module for additional applications cuts down on programming, debugging, and compilation time.

4. Allowing applications to share a module reduces memory requirements.
5. Having only one copy of a module ensures that the module can be replaced with a new version from time to time without having to worry that an undiscovered copy of an older version might still be lurking around somewhere in the system.

The fact that a routine only has to be coded once usually more than compensates for the extra effort that may have to go into generalizing the routine. The more often it's used, the more time you can afford to spend improving it.

SYSTEM SOFTWARE

Functions which are so general as to be of value to every user of the computer, such as I/O routines, sort utilities, file systems, and a whole host of other utilities, are usually included in the system software supplied by the hardware vendor. Just what facilities are provided, how sophisticated those facilities are, and whether the vendor Charges anything extra for them, is a matter of perceived user need and marketing strategy. Sometimes vendors choose to provide text editors and other development tools, and sometimes they don't. Sometimes they provide a very powerful database management system, sometime only rudimentary file access commands. And so on.

When hardware vendors fail to provide some needed piece of software, it may be worthwhile for the user to write it himself. If the need is general enough, software vendors may rush in to fill the void; or perhaps user pressure will eventually convince hardware vendors to implement it themselves.

In this way, many alternative products may become available, and the user will have to evaluae which approach he wishes to take advantage of, based on such factors as cost, efficiency, other performance criteria, flexibility of operation, compatibility with existing software, and the comparative benefits of using each product.

PRINCIPLES OF GOOD SYSTEM DESIGN

In case you may need to design your own supporting software, or evaluate some that is commercially available, let's summarize the techniques which will permit you to achieve the greatest degree of data, program, and device independence. I have already given illustrations of most of the following principles:

1. *Modularity* — Conceptually break everything up into the smallest modules you feel comfortable dealing with.

2. *Factoring* — Whenever a functional unit appears in more than one location, investigate whether it is feasible to "factor it out" as a separate module (this is analogous to rewriting A*B+A*C+A*D as A*(B+C+D) in math).

3. *Critical Sections* — Refrain from separating modules which are intricately interconnected or subdividing existing modules which are logically intact.

4. *Independence* — Strive to make every module self-contained and independent of every external factor except as represented by predefined parameters.

5. *Interfacing* — Keep to a minimum the amount of communication required between modules; provide a consistent method of passing parameters; make the interface sufficiently general to allow for later extensions.

6. *Isolation* — Isolate all but the lowest-level modules from all hardware considerations and physical data characteristics.

7. *Testing* — Test each individual module by itself as soon as it is completed and as it is integrated with other modules.

8. *Generalization* — Produce modules which solve the problem in a general way instead of dealing with specific cases. Be careful, however, not to overgeneralize. Trying to make a new technology fit the mold of an existing one may seem like the best modular approach, and the easiest to implement, but the very features for which the new technology has been introduced must not become lost in the process.

EXAMPLE — When CRTs were first attached to computers they were treated as teletypes, a class of I/O devices incompatible with two of the CRT's most useful features: cursor-addressing and the ability to type over existing characters. Putting the CRT in block-mode and treating it as a fixed-length file represents the opposite extreme: the interactive capalities are suppressed and the CRT becomes little more than a batch input device, a super-card-reader in effect.

9. *Standardization* — Develop a set of sound programming standards including structured programming methods, and insist that each module be coded in strict compliance with those standards.

10. *Evaluation* — Once the functional characteristics have been achieved, use available performance measurement methods to determine the areas which most need to be further optimized.

11. *Piecewise Refinement* — Continue to make improvements, one module at a time, concentrating on those with the largest potential for improving system performance, user acceptance, and/or functional capabilities.

12. *Binding* — For greater flexibility and independence, postpone binding of variables; for greater efficiency of execution, do the opposite; pre-bind constants at the earliest possible stage.

BINDING

As the name suggests, "binding" is the process of tying together all the various elements which make up

an executing program. Binding occurs in several different stages ultimately making procedures and data accessible to one another.

For example, the various statements in an application program are bound together in an object module when the source program is compiled. Similarly, the various data items comprising an IMAGE database become bound into a fixed structure when the root file is created. A third case of binding involves the passing of parameters between separately compiled modules.

Remember that at the hardware level, where everything is actually accomplished, individual instructions refer to data elements and to other instructions by their location in meory. The "address" of these elements must either be built into the object code at the time a program is compiled, be placed there sometime *prior* to execution, or be provided *during* execution. Likewise, information governing the flow of logic can be built into the program originally, placed in a file which the program accesses, passed as a parameter when the program is initiated, or provided through user interaction during execution.

Binding sets in concrete a particular choice of options to the exclusion of all other alternatives. Delayed binding therefore provides more flexibility, while early binding provides greater efficiency. Binding during execution time can be especially powerful but at the same time potentially critical to system performance. In general, variables should be bound as early as possible unless you specifically plan to take advantage of leaving them unbound, in which case you should delay binding as long as it proves beneficial and can still be afforded. Incidentally, on the HP3000, address resolution between separately-compiled modules will occur during program preparation (PREP) except for routines in the segmented library, which will be resolved in connection with program initiation. If your program pauses initially each time you run it, this run-time binding is the probable cause.

A SPECIFIC APPLICATION

About five years ago, we were faced with the problem of developing a system of about 300 on-line application programs for a client with no previous computer experience. Their objective was to completely automate all record-keeping, paper-flow, analysis, and decision making, from sales and engineering to inventory and manufacturing to payroll and accounting. The client had ordered an HP3000 with 256K bytes of memory and had already purchased about 20 Lear-Sigler ADM-1 CRTs. About 12 terminals were to be in use during normal business hours for continuous interactive data entry; the remaining eight terminals were primarily intended for inquiry and remote reporting. Up-to-date information had to be on-line at all times using formatted screens at every work station. Operator satisfaction was also a high priority, with two- to five-second response time considered intolerable.

DISCUSSION QUESTIONS

Based on the "principles of good system design" summarized earlier, what recommendations would you have made to the development team?

At the time, HP's Data Entry Language (DEL) seemed to be the only formatted screen handler available on the HP3000. Consultation with DEL users convinced us it was rather awkward to use and exhibited very poor response time. Also it did not support non-HP character-mode terminals.

We elected to write a simple character-mode terminal interface, which was soon expanded to provide internal editing of data fields, and later enhanced to handle background forms. We presently market this product under the name TERMINAL/3000. You've probably heard of it.

The compact SPL routines reside in the system SL and are shared by all programs. The subroutine which interfaces directly with the terminals is table-driven to ensure device-independence. By implementing additional tables of escape sequences, we have added support for more than a dozen different types of terminals besides the original ADM-1's.

If we were faced with a similar task today, would your recommendations be any different?

After completing most of the project, we did what should have been done much earlier: we implemented a CRT forms editor and COBOL program generator which together automate the process of writing formatted-screen data entry programs utilizing TERMINAL/3000. We call this approach "results-oriented systems development"; the package is called ADEPT/3000. Programs which previously took a week to develop can now be produced in only half a day.

Since we were using computers to eliminate monotonous tasks and improve productivity for our clients, it was only natural that we should consider using computers to reduce monotony and increase productivity in our own business, the business of writing application programs. If you write application programs or manage people who do, you also may wish to take advantage of this approach.

What features of VIEW/3000 would have made it unsuitable for this particular situation?

- not available five years ago
- HP2640 series of terminals only
- block-mode only (not interactive field-by-field)
- requires huge buffers (not enough memory available)
- response time and overall system performance inadequate

From what you know of TERMINAL/3000 and ADEPT/3000, how do these products enable a programmer to conform to the principles of good system design?

TERMINAL/3000 itself: modular, well-factored, single critical section, device-independent, independent of external formats, simple 1-parameter interface, table-driven hardware isolation, well-tested, generalized, optimized for efficiency, run-time binding of cursor-positioning and edit characteristics.

ADEPT/3000: produces COBOL source programs that are modular, well-segmented, device-independent, and contain pre-debugged logic conforming to user-tailored programming standards; built-in interfaces to TERMINAL/3000 and IMAGE/3000 (or KSAM/3000) isolate the programs from hardware considerations and provide device and data independence.

BIBLIOGRAPHY

Boyes, Rodney L., *Introduction to Electronic Computing: A Management Approach* (New York: John Wiley and Sons, Inc., 1971).

Hellerman, Herbert, *Digital Computer System Principles* (New York: McGraw-Hill Book Co., Inc., 1967).

Knuth, Donald E., *The Art of Computer Programming* (Reading, Mass.: Addison-Wesley Publishing Company, 1968).

Swallow, Kenneth P., *Elements of Computer Programming* (New York: Holt, Rinhart and Winston, Inc., 1965).

Weiss, Eric A. (ed.), *Computer Usage Fundamentals* (New York: McGraw-Hill Book Co., Inc., 1969).

Selecting Application Software and Software Suppliers

Steven J. Dennis
Smith, Dennis & Gaylord

We begin by looking at some of the data regarding the assumptions that have evolved regarding standard software packages and the software package industry.

THE MYTHS

Following are some of the myths — beliefs ("beliefs" are assumptions which may or may not reflect reality) — which have evolved over the past decade or so of application packages evolution:

- *Myth:* We can safely go ahead with our hardware purchase since there MUST be plenty of good software systems available.
- *Fact:* There are surprisingly few firms nationally that have developed truly viable packages. And fewer still that will be able to meet YOUR requirements.

In fact of the literally, tens of thousands of software firms, barely a dozen have sales of more than $10,000,000 annually. And of these, the largest barely tops $35 million . . . hardly international giants!

- *Myth:* Since all packages are essentially the same, we will budget for the low priced one . . . that will help keep the costs low.
- *Fact:* Costs certainly aren't the most accurate indicator of how good the software is . . . but you don't typically put retread tires on a brand-new Mercedes. Quality software — a system which has the quality you would expect help manage your organization effectively — is going to be a little higher priced.

There are lots of factors which go into software price — and probably only about 25% of them have anything to do with writing the programs (more on this in Part III).

- *Myth:* One great thing about a package is that we can install quickly . . . get it running in a month . . . maybe get several packages running in a month or so. . . .
- *Fact:* For one thing, remember that quality software firms are busy too . . . their schedules may not fit exactly with yours.

For another *YOU have to learn the package BEFORE your users do. You should run parallel (or at least develop a good test case). Your objective should NOT be just to see if the package works – but does it work for YOU.*

Take your time . . . do it right! After all, your organization will probably spend many months deciding on the right hardware — surely you can allow an extra month or two to make sure the software is operating properly.

- *Myth:* Our organization can't be TOO different from everyone else — we should be able to fit into a STANDARD General Ledger . . . STANDARD Accounts Receivable . . . a STANDARD Order Management System . . . a STANDARD . . .
- *Fact:* Data Processing should work for YOU . . . not the other around. Companies ARE different. Packages ARE different. Approaches ARE different. Features ARE different . . . and some are critical. A large multi-national firm doesn't just go changing its chart of accounts because THAT'S THE WAY THE SYSTEM WORKS!!!!!!!!
- *Myth:* Most applications are easy, straight-forward systems.
- *Fact:* Anyone who feels this way needs to design and implement from scratch just ONE Payroll system . . . just one INTEGRATED General Ledger to realize that there are NO easy applications.

If you find yourself saying to the software or hardware firms you are talking to, "We just need a standard A/P system . . ." catch yourself and re-think. You may be identifying yourself as the classic "easy mark." And easy marks have a way of giving their money and time to people who give little in return.

- *Myth:* The terms General Ledger, Accounts Receivable, Accounts Payable, Purchasing, Materials Handling, Resources Requirements Planning, Financial Modeling, and Inventory Control are universal . . . they mean the same thing to everyone.
- *Fact:* While your technical staff may think that General Ledge was probably just promoted from Colonel, ONLY your financial staff will be able to determine the features that are needed. Be careful . . . don't assume anything.

General Ledger does NOT necessarily include financial statements. Accounts Receivable systems don't always let you apply cash to ANY account (not just to

---

This paper is an excerpt from the book *A Success Plan . . . for Software/Implementation* by Steven J. Dennis and Barry Barnes. Published by Barry Barnes & Company, 1982.

an open receivable). Order Management differs radically for manufacturers vs. distributors.

Many systems do not allow for much FLEXIBILITY in your chart of accounts. Few packages allow you to customize the software to your organizations's own UNIQUE requirements (short of an almost complete rewrite of the software).

- *Myth:* A package is a system that is operating successfully at some other company . . .
- *Fact:* Wrong! Packages are WRITTEN to be PACKAGES!!! They are not — NOT — custom systems which happen to work at an organization which may be in the same industry as yours.

And, remember a package is also *not* something that is SCHEDULED to be done . . . it is something that IS done.

A TRUE software package is one *developed to be a package* by a firm that has *as its business* developing and supporting PACKAGES!

- *Myth:* All software packages obviously contain the proper audit trails and accountancy . . .
- *Fact:* Make sure your controller, chief financial officer, administrator, vice president of finance, business manager, or your CPA takes a good look at the package . . . we have heard of too many audits that insisted on changes to existing software . . . expensive changes . . . to include fundamental audit trails (the ones everybody *assumed* were there in the first place).

The people who write software don't always understand accountancy considerations . . . on the other hand, just because the firm has strong accounting credentials, doesn't mean that their software adheres. ASK about controls, audit trails, security, and the philosphical underpining of the software products.

- *Myth:* Just because you have a computer or are getting a computer . . . just because management feels that EVERYTHING should be automated . . . just because there are relatively inexpensive packages crying to be bought, we should surely bring ALL applications in-house.
- *Fact:* Some applications require careful analysis before a final decision is made to go in-house. Payroll is the best example: For small companies we have often recommended against the hassels of maintaining their own payroll . . . changes in tables, government forms, minimum wages, unions, etc., require a heavy in-house investment . . . well worth it for many organizations — but definitely not for everyone . . .

Be appropriate. Automate when there's some definable distinct advantage to the organization. Automate when you expect to be able to see *results*.

- *Myth:* For those of you whose organizations have an in-house data processing staff, "there just isn't ANYONE out there who can develop a system better than we can right here in our own organization" (also known affectionately as the "Not Invented Here" syndrome . . . or, often, more accurately, as the "Kiss of Death" or "Results Not Yet In" syndrome).
- *Fact:* First off, you are probably viewing it from the *technical* side . . . and from that viewpoint there may well be some truth. After all who better knows the DP philosophy, particular hardware configuration, internal politics, etc., better than your own data processing department.

In fact, the software house you select should be an expert in PARTICULAR applications — they know General Ledger, Order Management, Payroll, Medical Billing, Financial Modeling . . . and in the long run THAT's what you need.

And, by the way, (to the surprise of the DP staff) a good software package will often be impressive technically as well). This is much truer now than in the past. "Mature" packages are often relatively new — and often written using software development tools that simply weren't available a few years ago.

- *Myth:* With a wealth of new systems and languages, programming is now much easier than before . . . surely I can write my own applications.
- *Fact:* Programming is not so terribly difficult . . . but desig a particular function — or an entire system — requires the experts. The main benefit of the advent of powerful software development tools is that they free up time to do a more comprehensive job of DESIGN and that's fundamentally why packages are so suddenly such a viable alternative.

Learning to speak another language is one thing . . . writing a novel using this new language is quite another! And, writing that novel error-free? . . .

Programming is only approxiamtely one-sixth of the total effort . . . the whole picture looks something like:

- Design ..... one-third
- Programming ..... one-sixth
- Test/Debug ..... one-third
- Training & Documentation ... one-sixth

- *Myth:* We MUST have our programs written in COBOL, or some other such language.
- *Fact:* Arguing for a particular computer language is usually ridiculous. It's like arguing for French, Spanish or German — all of which are excellent languages.

Properly used — FORTRAN, RPG, COBOL, BASIC, and many other user-oriented high-level languages can provide *excellent* solutions.

Remember, even though *you* may think English or Danish, or whatever is the world's best language, millions speak others . . . write others . . . get results in others.

- *Myth:* Choice of language doesn't matter AT ALL . . . choice of file handling technique doesn't matter AT ALL . . . as long as it WORKS!!!!
- *Fact:* Buying something completely non-standard

can be a disaster . . . insist on complete documentation for anything that looks a little out of the ordinary.

• *Myth:* Since my company deals with a single-person (or small) law firm . . . or a single-person or small CPA firm, it's okay to get my software from a very small software house . . . or even from an individual.

• *Fact:* This is a tough one . . . certainly there are many, many excellent, well-qualified software firms. Remember, though, our industry does not yet have a bar exam or any accepted professional certification like the CPA . . . nor are there ANY levels of standards throughout the industry which compare with the standard "generally accepted" accounting practices that all CPA's follow.

On the other hand, if your small (one, two . . . five-person firm) KNOWS their business — and yours — they *may well be far superior* to the 100-person company which views you as a small fish in a big pond. We know of a company — one of the 10 largest in the world — which actually PREFERS to work with small (one to ten-person) software firms.

Put this test to work . . . if my CPA went out of business, where would I be? Probably all right — there are others who could step right in. NOW if my software consultant/supplier went out of business then what???

• *Myth:* There are software firms to whom I can turn over complete responsibility for my implementation . . . total *turnkey* solutions . . . software houses which will sign a contract guaranteeing success . . . with penalty clauses . . .

• *Fact:* There ARE firms which will tell you that they'll take on all responsibilities. But let's face it . . . whose system is this? Theirs? Unless *you* take the responsibility for the solution to work . . . invest the time . . . invest the energy . . . adopt the right attitude . . . you will be developing all the ingredients for failure.

And, remember, desperate people will sign *anything*. A firm that knows how to do business doesn't have to sign your attorney's document in blood — they'll drop you like a hot potato and move on to someone else who knows how to get results.

• *Myth:* Custom software is never necessary or if it is, it should always be done in-house.

• *Fact:* Not true! Custom software IS quite often required.

For example: We have worked with a large client which fabricates and erects steel for many of the really *large*, modern high-rise office buildings and hotels in the Western U.S.A. Recently, a custom system was developed (by an outside firm) for this steel fabricator — one that estimates, to the nut and bolt level, the steel requirements for a 40-story office building . . . determines the best source throughout the world for that steel . . . how to transport it . . . and how long it'll take . . . and cost. That doesn't exactly lend itself to a package.

SUMMARY

There is an emerging — definite — context for standard application software packges. The degree to which workable solutions and procedures evolve for the successful incorporation of this new field into our business life directly affects the results we can expect for the near future.

The use of the data outlined in this presentation can assist in *initiating* the process of getting RESULTS . . . for your organization and for others.

Now, let's move on to an examination of the process of assessing software, selecting the software supplier, and implementing the software system.

PART II
SELECTING A SOFTWARE SUPPLIER

INTRODUCTION

The approach outlined in this section of this book is most appropriate for companies in the $10,000,000 to $10 Billion annual revenues categories. Smaller companies tend to be able to fit extremely well into totally "standard" packages — often being able to change their mode of operation to fit the package. Larger organizations — particularly when they hit to $25,000,000/year level — tend to have developed unique operating styles, unbendable procedures, inflexible managerial requirements, or just plain strong preferences.

In general, very small organizations can ignore a lot of what we propose in this book. Yet . . . the fundamental underpinning of RESPONSIBILITY for results — a commitment to being successful — will still serve well!

CONTEXT — A VARIETY OF VARIABLES

Packaged software has become the major area of focus in the information systems field. Yet, the industry called "Packaged Software Firms" has no equivalent of a FORTUNE 500. In fact, only recently has the situation arisen where there are more than a dozen companies in the world which have annualized sales of more than $10 million (and those few have only recently attained that level).

The vast majority of software firms — including the QUALITY ones — are small companies doing between one-half million and five or six million dollars per annum. And, small businesses are sometimes subject to radical ups and downs.

Thus, the selection of a software firm needs to be based on a set of criteria which optimize the potential for success. It may well be the case that the *firm* supplying the software is as important — or more so — than the software itself.

The important point to know . . . and acknowledge . . . and accept — whe you like it or not — is this: In selecting a software supplier, you are establishing a *long-term business relationship!* And, if you do your job WELL, you'll establish a really long-term relationship.

So DO your job well. Exercise the same care you'd use in selecting your CPA firm and your Corporate attorneys.

PURPOSE — A TOOL FOR MEASUREMENT

This set of guidelines encompasses what we've learned from our own experience as consultants and as software suppliers. It is developed from a variety of viewpoints. We offer it as a tool to use to measure any software firm which offers standard application software packages.

GUIDELINES — CHECKLISTS & PROCEDURES

Following are a series of "bullet-item" guidelines. These can be greatly expanded: These lists are by no means meant to be all-inclusive . . . they're simply a good start.

Give the software suppliers you deal with an opportunity to present their story to you — in their own way; then use these items as a checklist.

Don't expect any one firm to get an A+ on all items. We *know* that we're not "There" on them all . . . and we probably never will be! A score of 70 to 80% is "Excellent"; 90%, "Superior"; 100% . . . well . . . come on . . .

THE SELECTION COMMITTEE

Application software packages should NEVER be purchased by a single person . . . and that's not an indictment that *individuals* can't adequately make the decision. Some can.

The truth is: Software must be implemented by a variety of people at different levels in different functions within the organization. The wise organization will get a variety of viewpoints from the start.

In general, the Committee Model doesn't work in this world (as governments strive ernestly — and repeatedly — to prove). Yet, here is an example where a team of well-chosen effective people can make a real significant contribution.

Ideally, the Selection Committee should include representatives of the following functions . . .

- Organizational Management
- Functional Management
- User/Operator
- Data Processing/Technical
- System Implementor — the person who will have the responsibili of successfully implementing the system once it's chosen.
- System Coordinator.

A properly selected (and operating) Selection Committee can achieve tremendous results for the organization. It should meet frequently during the buying cycle. Each member should diligently fulfill assigned responsibilities.

THE SOFTWARE/SYSTEM ACQUISITION PROCESS OVERVIEW

We can't say what works for everybody . . . but in our experience as managers, users, consultants and software suppliers, we've found the following procedure to be a valid one:

Organizing Yourselves

- Define the Selection Committee
- Define the overall, broadbrush implementational schedule
- Develop a *brief* Selection Committee charge and guidelin
- Develop a *brief* background and requirements document fo the software suppliers

The Review Process

- Define the software packages to evaluate
- Poll your colleagues for additional ones
- Call your friends
- Ask around at social occasions and cocktail parties
- Poll the computer vendors for others
- Contact your CPA (and other consultants)
- Preview the various directories
- Define acceptable computer vendors
- Collect literature & documentation
- Contact the software suppliers
- Contact the hardware vendors

Our recommendation is an unusual one. We recommend that you do the front-end work yourself (previewing brochures, telephoning software suppliers, calling a few of their resources), and quickly narrow it down to three to five finalists which you'll then *visit* . . . and then send your RFP (if you use one) to only those. It saves you a lot of time and energy in the long run, while letting you be sure that the supplier who responds with a proposal actually knows something about your business.

The Preliminary Evaluation Process

- Develop a preliminary set of selection criteria for the software packages (Software Requirements Checklist)
- Develop a preliminary set of selection criteria for the software firm
- Find out more about the software firm (by telephone)
- Find out more about the application packages
- Procure the software supplier's client lists
- Telephone the software firm's references
- Select three to five (3-5) finalist firms
- Visit the finalist software firms
- Visit the computer vendors

The Interim Evaluation Process

- Analyze the findings of the visits
- Expand the Software Requirements Checklist
- Adopt the budget for software, hardware, etc.
- Develop the Request for Proposals (if appropriate)

The Final Evaluation Process

- Receive and review the proposals
- Prepare the Comparative Requirements Analysis
- Review the responses (entire Selection Committee)
- Review the responses with your users
- Review the responses with the software suppliers
- Select and advise the software supplier

The next step may be the most important part of the process. Make s you get to know ALL the people with whom you'll be working . . . and that they get to know — and like — all your key people. This is a people-oriented business. The better you know, and understand each other, the better will be the overall level of affinity and communication and reality when the going gets tough . . . and it *will*, at one time or another, get tough!

Establishing the Client/Software Firm Relationship

- Complete the financial requirements & procedures
- Complete the legal/contractual requirements
- Establish the technical support contact points
- Establish the software training procedures
- Establish the documentation update procedures
- Establish the user support procedures

In general, it's the hardware vendor who will maintain the computer equipment and the operating system. DON'T rely on the sales representative . . . he or she has other sales to bring in! Get to know the people who will support you.

Establishing the Hardware Vendor Relationship

- Establish the relationship with the hardware vendor's operating software and hardware maintenance staff (SE's & CE's)
- Complete the legal and financial requirements with the hardware vendor
- Issue a purchase order for the required computer hardware and operating software

Implementation Planning

- Define the Implementation Review Committee (may be the same as the Selection Committee)
- Adopt a requested Implementation schedule
- Present the requested schedule to the software supplier for review and resolution
- Review the software supplier's recommended Implementation Plan
- Mutually resolve discrepancies to achieve an Adopted Implementation Plan

General Application Preparation

- Schedule applications training
- Schedule technical training (if appropriate)
- Review procedures for potential modification
- Conduct weekly or bi-weekly Implementation Committee review sessions
- Procure the software supplier's final recommendation of the hardware configuration
- Finalize the hardware configuration

General Computer Hardware Preparation

- Conduct the site review for the computer
- Prepare the computer site as appropriate
- Analyze & define CRT and hardcopy printer locations
- Arrange for cabling and modem installation
- Conduct Implementation Committee review meetings

Final Implementation Preparations

- Initiate applications training
- Initiate hardware training
- Initiate other technical training
- Conduct project activities for special/custom work
- Conduct Implementation Committee review meeting

Implementation

- Install the computer
- Install the software module(s)
- Load/convert data to the new software
- Conduct application testing
- Conduct Implementation Committee review meeting
- Conduct end-user operational training
- Implement the application(s) on a "live" basis

On-going Review

- Conduct periodic review sessions with end-users
- Conduct periodic Implementation Committee review meetings
- Conduct periodic software firm review meetings
- Conduct periodic hardware vendor review meetings

This checklist is *one* way to acquire software. It's surely not the ONLY one . . . it just works! Try it . . . or modify it. But whatever you do, have a commitment to get results; develop a a Plan . . . then WORK the plan.

THE SOFTWARE REQUIREMENTS CHECKLIST (THE STATEMENT OF REQUIREMENTS)

The Software Requirements Checklist (also known as the Statement of Requirements) is the common thread for the *Functionality* of the software to be chosen. Most of what should go into this document is a list of *features* which you want or need . . . thus, it's impossible at this time to descri exactly what should be on it. Following, though, is a set of guidelines for preparing it:

- *Don't overlook the obvious* . . . don't "assume" that what want will automatically be in all packages.
- *Clearly state your requirements.* Avoid lots of text . . . it doesn't get read; use lists or checklists, where possible.
- *Rank your requirements.* Don't get too fancy . . . "A" for Must-haves; "B" or "3" for Highly-Desirable; "C" or "2" for Nice-to-Haves or Tie breakers.
- *Include philosophical or approach requirements* — things like on-line vs. batch . . . language . . . options . . . decentralized vs. centralized management style . . . growth . . . plans . . . security requirements . . . accounting batches . . . auditability . . .
- *Include interface or customization requirements*
- *Ask about product expansion* — are updates provided? How How often? What happens if you customize? Interface? Is a software support agreement available?
- *Ask about documentation* — what's provided? How readable is it? Who writes it? How often is it updated? Do you *get* it? How many levels of documentation are there?
- *Ask about training* — what's provided? How often? Who attends? Are there standard classes? Do you get the training materials? Who conducts the training? Who attends the training?
- *Ask about installation procedures* — what checklists wil you have? What procedures will be provided? What assistance will you get?
- *Find out about support* — what happens when you get in trouble? How do you report software bugs? What facilities exist for phone consultation? How often is user documentation updated? What procedures exist for follow-up on your requests? How do you suggest changes and the "wouldn't it be nice . . .," enhancements or extensions? Does the firm have a standard support program and a standard support contract?
- *Describe your organization* — tell the software firm abo your objectives (lists — not narrative); get each functional unit's requirements; get alignment so that you'll all be using the same terms . . . expecting the same results.

A well-designed Software Requirements Checklist should have terse one- and two-line features/requirements statements with a place to the right (or left) for the software supplier to enter a short yes/no/"*" response. Each *grouping* of features should be followed by a "*" blank space to write responses, exceptions, n rates, quotations for custom work, future release dates, etc. In other words, make it EASY to USE.

WHAT ABOUT CUSTOMIZATION?

No matter how much you may desire otherwise, your application may well have requirements that just aren't available from a standard package. We've seen requests for such applications as railcar tracking, event scheduling, loan tracking, event-driven action item management, and such — applications which are mainstream to the company, and which must reflect the *specific* approach of the organization.

The need for customization need not be a catastrophe . . . IF . . . you understand the implications. Learn how to define the need:

- Check against the Statement of Requirements for *alternative approaches.*
- Use the software firms' *consultative assistance* — they have suggestions as to how others have solved the same problem.
- Seek out *parameter-driven software* — it may be tailorab your need.
- *Re-evaluate* your need . . . if there's no package availa just may be that you're doing things too differently.
- Don't be afraid to *stick to your requirements* . . . perh organization's way — non-standard as it is — is the one which gets results!
- Determine whether standard package modifications required a *structural or are general enhancements* . . . that is . . . are you adding a room . . . or repainting . . . do you need a new foundation. IF the changes are structural — go custom (or buy the package to use as a building block for a custom system, with the understanding that you must take ownership of the resultant system).
- If you've found a "fit" or a near-fit for other applicati *discuss with the software firm* their interest in developing your custom requirements . . . or their recommendations . . . or how cooperatively they'd work with another firm which you might engage.

Most software requirements can be met from a package . . . but NOT ALL . . . not even all the "of course THAT would be in a standard package" requirements for a given function may be in the package you like best (and it may still be the best package to buy).

THE SOFTWARE DEMONSTRATION

Seeing the software work is extremely important. Software — like the people who design and use it — has a personality. Ask for a software demonstration!

The demonstration is best done at the software firm itself (assuming it has its own computer).

- Define in advance which modules you want to see.
- Send the software firm a copy of your Statement of Requirem in advance.
- Get the RIGHT people there.
- Come prepared.
- Give the software firm an extra hour or two to tell their story.
- Take the software firm's advice.
- Allow sufficient time.
- Ask for an extra hour or so — at the end of the session to unanswered questions handled.
- Tell what you expect to accomplish.
- Keep an Open Mind!
- Keep on Track.
- Be courteous.
- Ask Questions.
- Look at the Audit Trails.
- Look at the User Documentation.
- Look at the Technical Documentation.
- Look at the human engineering.
- Expect to have a reasonably good fit — (75% to 85% is considered a good fit; 90% is considered excellent; 100 is rare!).
- Expect to find gaps.
- Beware of the Everything's Great Syndrome.
- Be professional.
- Meet the user support staff.
- Don't demand to spend lots of time with the technical staff
- Follow up on Unanswered questions.

A good software demonstration can enlighten you as to omitted items on your Statement of Requirements. Also, it's an opportunity to interact with key people in the software firm who may end up being your contacts for years to come.

Use the demo effectively . . . then go back home and update your Statement of Requirements. *Have a Selection Committee meeting after each such visit.*

THE REQUEST FOR PROPOSALS

This is probably the area in which the *most* mistakes are made . . . by the requestor!

In the first place, use an RFP where it's *appropriate*. If you find a package during the preliminary search which meets your needs . . . offered by a firm that fits all the selection criteria, then, WHY go through the misery of an RFP? And, the RFP is as much work for you as for the software firm (if you're doing it properly).

Admittedly, it is a controversial stance to recommend *against* RFP's . . . but more TIME — and, often, MONEY — is spent by some prospective buyers, going through the drudgery (and motions) of Proposal Requests than is often spent on *acquiring and implementing* the software itself!

Let's look in more detail at the *purpose* of the Request for Proposals.

What an RFP Isn't

- It is NOT a way of justifying an already-made decision.
- It isn't a guarantee (those are up to YOU!)
- It isn't a fancy way of covering shoddy selection processes.
- It isn't a "test" for the software supplier (quality soft firms throw away the ones which look like "tests").
- It isn't a way of badgering others into doing things your way.
- It isn't a document more than ⅛" to ½" thick (anything thicker is a request for free consulting services).
- It is NOT something to be tied into a contract (most software firms which will agree to tie the RFP and their response to the contract aren't capable of living up to a court challenge; who *can* deliver what they say aren't interested in complicating their legal agreements . . . on strong counsel of their own attorneys!).
- It isn't, in summary, much more than a statement of requirements . . . than a statement of requirements . . . a Statement of Requirements.

What an RFP IS

- A Statement of Requirements.
- An opportunity for the software firm to tell its story (in *own* way).
- A place for summarization of costs.
- A place for summarization of potential implementation sched dates and events.

Guidelines for the RFP

- Make it *friendly!*
- Make sure you *personally* contact the firms you send it REPEAT: Personally telephone — or, better yet, VISIT — the software suppliers you want to respond.
- Send it to NO MORE than *three to five* firms.
- Tell about your organization (briefly).
- Give a brief background of the situation for which you're seeking solutions.
- Include the Statement of Requirements.

- Allow the software firm to answer in its own style.
- Attach an outline of what you want to know about the softwa firm. Keep it to the "need to know" level.
- Avoid rigid formatting requirements.
- Don't demand all sorts of contractual modifications . . . suppliers just aren't interested in doing business that way (no matter what counsel you get otherwise).
- Don't ask the supplier to tie their response to a contract good firms spend enough time on their contractual agreements to do business well — and their attorneys counsel them heavily against such modifications.

On the other hand, if the software firm has shabby-looking or weak contracts, you're entitled to ask for contractual modifications . . . but . . . do you really want to do business with such a firm?

- Remember: The software firm is busy — particularly if they're competent. If you ask for two weeks worth of work to respond to an RFP, you'll only get the software suppliers who aren't in demand.
- Get competent assistance in evaluating the responses.
- READ and ANALYZE the responses.

As a summation, regardless of whether you agree with us on the value of an RFP, don't let the RFP be a substitute for human interaction, client reference-checking, . . . software firm visits . . . and just plain hard work. If it's worth doing, it's worth spending some time doing right.

EVALUATING THE SOFTWARE FIRM

It is almost as — maybe even more — important to carefully assess the software supplier as the package itself.

REMEMBER: You are establishing a long-term business relationship. Just because this is a highly technical field, don't be bamboozled into anything. Look for the *same* organizational attributes that you'd look for in ANY company!

The successful software organization has to know their business . . . and must be committed to supporting clients using standard software products.

Following are some checklist items we've come across. Perhaps you can extract several items and develop a weighting/evaluating schema which will work for you.

- *Look at Outward Appearances:* How does the software firm va itself. What evidence is there that they've been around for a while . . . survived the magical "New Company Mortality" syndrome? How have they invested in the future? Look for "gut-level" feeling. DO they *look* like winners? Check these attributes:
  - Facilities
  - Equipment
  - Furnishings
  - Clients
  - Checklists & Procedures
  - Documentation
  - How the people view things (values)
  - Organizational Structure
- *Look at the People who Create the Packages:* Again, use gut-level approach. Would you want them working for your organization?
- *Look at the Firm's Background:* More gut-level stuff. Lo them and their experiences . . . and don't assume there's any one "right" way. Listen carefully to their story.
- *Look at the Knowledge Level of the People:* How have the all together? Does the firm have what it takes to succeed in ALL its areas (not just the technical)?
- *Look at their Guidelines and Standards:* Examine the evi of commitment. *Written* standards are statements of intention, stability, permanence. They're important!
- *Look at their orientation:* This is incredibily important . . . this is the firm's *real* purpose. Ask candid, yet open, friendly questions.
- *Look at the Firm's Target Marketplace(s)*
- *Check for Internal Procedures:* These are the indicators attention to detail. And, that's critical in the software field.
- *Look at the Support Apparatus:* This will be your contac AFTER the sale. You only interface with the sales or business development function BEFORE you sign up . . . IF you select the right firm.
- *Look at What Goes Into Product Price:* The difference in price is immense. There are, in many cases, equally-developed software packages available from different firms at vastly differing prices. What's the difference between a $3,000 Order Processing system and a $40,000, $50,000, or even a $100,000 one? Generally, there ARE features differences . . . but not always. The difference may well be in the firm. How does a firm selling their product for ten times that of another one of equal "features" survive? Generally, because they appeal to organizations which *value* long-term commitments.
- *Look at the Software Firm's Sales Style:* Here's where a of the true values of a firm show up . . . to the extremes. If the sales representatives act like a stereotyped salesperson, then there's probably something behind the scenes which supports such an approach. On the other hand, if the people responsible for business development show good knowledge, experience, and a consultative approach which demonstrates genuine concern for your success, they're probably reflecting some

very solid and fundamental philosophies and policies of the firm . . . latch onto them.

- *Look at the Implementation Planning Assistance:* The understanding of implementational considerations is one of the most positive indicators of a real-world, results-oriented firm.

  If the firm begins talking implementation, listen to them. Chances are, if they've passed the other tests, they know far more about how to implement a system than you. After all, they're probably doing it between ten and a couple of hundred times per year!
- *Look at the Legal/Contractual Instruments:* If you get a double-spaced typewritten page for a contract, feel free to take a hatchet and an army of attorneys to it.

  On the other hand, if you get a typeset, well-structured document that provides for mutual protections and which incorporates and documents business procedures, then expect the firm to be relatively resistant to modifying it.
- *Look at the Firm's Growth:* The software industry is in explosive environment.

  Unfortunately, even organizations with shoddy products and shabby outlooks can survive — even grow rampantly! Growth is a tough thing to handle . . . and it's roughest on the firm which is committed to quality.
- *Look at the Company's Management Style:* Despite all the growth, the firms which will *truly* succeed (for themselves and for you) are the ones which MANAGE themselves well . . . just as in *any* field, good management pays off.
- *Look at the Company's Business Ethics*
- *Look at the Software Firm's References:* Ask for references . . . AND . . . then *contact* them.
- *Talk to the Software Firm* — This has to be the most imp criteria of all. Talk to the software firm as you would to ANYBODY who could truly assist you; don't worry about giving too much of yourself and your values to them . . . if they're unethical, they'll *definitely* try to take advantage of you — be mature enough to be willing to find that out beforehand.

  Choose the software firm as you would your CPA firm or corporate attorneys. Choose them using the same criteria you would if your organization were going to acquire them . . . or if you were going to invest in them.

AFTER YOU SELECT THE PACKAGE . . . THEN WHAT?

There are several steps which should be taken. The important thing is NOT to assume that you're "there" . . . indeed, the journey is still somewhat in its infancy. In fact, you're just beginning! There's some more internal organizational analysis that needs to be done . . .

- Determine what people-problems you will have with software tha cuts across organizational lines . . . Order Processing for example can affect marketing, sales, credit, customer service, production control, manufacturing, shipping, quality assurance, AND accounting . . . what will your *people* problems be?

  Develop a plan for handling the inevitable . . . it WILL occur!
- If you have to modify procedures to fit a selected package, try it *manually* first. Get the resistance out of the way . . . PRIOR to having the computer to blame.
- Get the user to sign off on the system . . . the Accounts Re supervisor will be much happier if he or she blesses the system *in advance*.
- Take ownership of the system . . . and make sure everybody — including management — expects results . . . and is committed to doing whatever it takes to GET results. Finger-pointing and blame and "reasons" just simply have no place in the implementation phase. If they crop up, acknowledge them for what they are (the things people do when they're NOT getting results) and MOVE ON! (to getting results).
- If you haven't already done it, list your required enhancem Have the software vendor quote/ recommend how these enhancements should be done.
- Develop workarounds for all the functions which aren't exac the way you'd like the package to work — and inform everybody, so there won't be the excuse of: "Well, this package just isn't the way we should be doing things."
- Make sure you get completely trained on the software (from perspectives: User . . . technical . . . and standards). Make certain the user is fully trained . . . that there's the *ability* and *willingness* to understand.
- BE PREPARED — remember that users CAN damage themselves thr no fault of the software house or the software.

  Or even: "Advised of a schedule change??? Call the dispatcher."

  Convert some or all of the members of the Selection Committee into an *Implementation Committee*.

  Identify ONE person (for each module) as the System Implementor — that one person who has the ability and the responsibility for getting results . . . and who is recognized and respected by others as able and willing to make it happen. Have a meeting of the key players at least once every two weeks.

  Once the selection has been made, there's the

cumbersome job of getting things rolling. And that's where the software firm's many experiences can assist you . . . that's where Implementation Planning and Project Control procedures come into play.

THE PURPOSES REVISITED

The purposes of this document are plain and simple: To provide at least *one* quality, honest, "what's so" step forward. Specifically, we offer a considerable amount of data, several methodologies or processes, and a sharing of experiences which may support you and your organization in attaining the successes you want. The only real value you can get from this book is a willingness to look at things *as they are* . . . followed by using whatever portions of our data, methods, and experiences which prove to be useful for you.

Office of the Future — Starting Today

Mark S. Trasko
Dynamic Information Systems Corporation
Denver, Colorado

INTRODUCTION

Through the past several decades, computer hardware and software have evolved and expanded to meet our various needs. Scientific data processing, process and manufacturing control, database management are three domains where computers are predominant. In the 80's, a fourth area of application will rapidly emerge: the realm of inter- and intra-company communication, often referred to simply as "office automation." Electronic mail, word processing, automated message systems and time management tools are some of the many functions of office automation systems that will increase productivity.

This paper surveys current activities in the office automation field. It then explores some future directions for this technology, and implementation potential on the HP3000. Finally, the paper focuses on a product, DATADEX/3000, intended to serve as a cornerstone of office automation systems.

THE PRESENT

Currently, office automation tools are available for most computer systems. These products provide a variety of functions, with emphasis in the following areas:

- Word processing, often with access to an on-line dictionary for spelling error detection and correction and word hyphenation.
- Electronic mail facilities to distribute documents that exist on the system (potentially a computer network). Most mail systems provide for verification that a document has been received, replies, the inclusion of comments prior to routing to further destinations, etc.
- Electronic memo systems which are similar in capabilities to the mail systems just described, except that text is generally brief, and is entered interactively by the sender. As with mail systems, most electronic memo facilities allow routing to multiple destinations, adding of comments, etc.
- Electronic scheduling. This includes personal time management tools and the scheduling of meetings and shared resources such as conference rooms and equipment.

The above office automation facilities are available on computer systems provided by several manufacturers, including IBM, DEC, Prime, Wang and others. As of January of 1982, Hewlett Packard had introduced word processing hardware and software. Given HP's publicly announced commitment to the "interactive office," it is likely that they will soon release additional tools in the office automation area. In addition, independent HP3000 software vendors are likely developing products aimed at this market which will soon be available.

Beyond the four categories of commonplace office automation tools just discussed, some more advanced products are available from two vendors that indicate possible future directions for office automation.

Xerox has the Star Work Station, an expensive but powerful system that allows the manipulation of documents and other textual information through the use of icons. Icons are graphic representations of either actual physical devices or physical equivalents of logical entities such as files. For example, a disc file might be represented as a file folder or letter, depending on whether the file contained one or several documents. A Star terminal includes a "mouse" to facilitate easy movement of the cursor among the several icons typically displayed on a screen. Placing the cursor on an icon and pushing a button on the mouse designates the device or entity associated with that icon as the source or destination in an operation.

One icon can father several other icons, similar to a menu screen. Thus, an operator can start with a view of the whole office, then focus all the way down to an individual character in one document. Since common visual associations are used, little training is required before an operator can command a wide variety of operations. However, the high cost of the system may have to be reduced before the savings in training costs justify its widespread usage.

A second advanced office automation facility has been announced by Wang. Called "DVX" (Digital Voice Exchange), it is an audio version of an electronic memo system. DVX is faster and for most people easier to use than textual memos, and may well be more effective since faithful reproduction of the speaker's voice is preserved. Wang is also developing a system to process digitized speech, allowing direct editing of speech with a fair degree of flexibility. However, word processors have far more flexible editing capabilities, so individuals that use a dictaphone almost exclusively and wish to

edit their own speech are the most likely candidates for this system.

The facilities just described are predominant areas of concentration in the office automation field today. Their use will grow rapidly throughout the next several years, and it is likely that HP or third party vendors will provide products in these areas to meet the needs of HP3000 users. The current state of the art in office automation emphasizes productivity and efficiency gains at two levels:

- Making clerical staff more effective.
- Reducing the time spent by management on communications "overhead."

For example, word processing systems provide significant time savings and productivity gains for clerical help, particularly when all or part of a document is used on more than one occasion. These systems also reduce the time spent by management and staff in the interaction required to finalize a new document, because editing, formatting, and printing operations can all be accomplished more quickly.

Similarly, electronic message systems and scheduling facilities reduce the time spent by management and staff in numerous non-productive but otherwise necessary activities. Message systems allow efficient communications within a company, helping to eliminate much of the "telephone tag game" that is usually prevalent. Scheduling systems can greatly reduce the time required for the iterative process of scheduling a meeting, especially if attendees maintain an on line schedule of times they are available.

Generalizations are almost always dangerous. However, viewing office automation as the implementation of various tools to increase the efficiency and reduce the cost of communication seems appropriate. Nearly all the office automation tools currently available deal with the generation, manipulation, or distribution of textual information. (Speech is grouped with written text, since they share a common purpose in office systems.) The one exception, electronic scheduling, still falls in the category of communication in the broad sense. Scheduling systems reduce the communication time required to arrange meetings, and meetings themselves are interactive communication!

THE NEED

Close examination of the current state of the art in office automation reveals a significant deficiency. Available systems deal primarily with "outbound" communications — textual information that will be disseminated. This certainly is a crucial function. All businesses must communicate information to other businesses. Large companies must also communicate information from office to office. And individuals in all but the smallest organizations often must communicate on other than a "live" basis.

However, these tools do not address the equal, or potentially greater, need to manage "inbound" communications. Most communication is bidirectional, with a high likelihood that as much or more information is received as is transmitted. Correspondence, legal documents, periodicals, books, data sheets, brochures, catalogs, all are examples of textual information that usually originates from outside the office area. This information is not only very important, but is received in large and ever increasing quantities. Yet current office automation tools can do little to support the management of this information. They are addressing only half of the need.

What is meant by "manage information?" The implications are the same as in the phrase "database management system." A DBMS organizes data so that a desired subset of the data can be retrieved quickly when required. The larger the database, the greater the need for efficient retrieval capabilities. Similarly, the more textual information a company must manage, the more important rapid access to that information becomes.

Even if the information has been internally generated, or has been received by electronic mail and thus can be stored on-line, the difficulty in managing it is nearly as great as with hard copy documents received from outside sources. Whether the information resides in computer files or file cabinets makes little difference. The key issue that must be addressed is: Can needed information be retrieved quickly and easily, or must exhaustive searches of all documents be performed to retrieve the ones desired?

Unfortunately, database managers such as IMAGE cannot by themselves meet this need. Textual information is excellent for communication, but poorly suited for storage and retrieval using conventional database managers. Text is unstructured and free format, whereas database managers are optimized for storage and retrieval of structured, formatted data. A new strategy is required if information of a textual nature is to be managed.

Before considering such a strategy, a pertinent question should be answered: How great is the need to manage such information? The ability to harness the wealth of information that a company receives on a daily basis, to eliminate duplication of effort in creating, maintaining, and retrieving such information, will provide a company with a significant competitive edge.

Information is important at all levels of a company. Executives and managers must have access to a wide range of information so that they can base decisions on the most accurate, up to date information that exists. The efficiency and productivity of staff personnel depends on fast, flexible access to information pertinent to their activities and responsibilities. Technical professionals function in a constantly changing technological environment. The quality and competitive standing of products that they design are impacted by their ability to access state of the art information in their fields. Successful support of delivered products is heavily de-

pendent on the availability of up to date product information to personnel in the field.

THE INFORMATION BASE CONCEPT

Meeting the needs of those who must have access to information can be termed Information Base Management. Facilities that support this function will be a key cornerstone to future office systems. The ideal information base provides fast, flexible access to the wide range of information vital to a company's operation.

As noted earlier, the scope of information that a company must deal with is very broad. Most of it comes from outside the immediate office area. Periodicals, letters, brochures, data sheets, catalogs, many legal documents, all are examples of such information. Documents that were not created internally or received through electronic mail cannot be maintained on-line, but instead must be physically filed. Again, whether a document is physically or electronically filed matters very little. The crucial issue is whether or not documents containing the required information can quickly be located among the huge store of textual information that a company must maintain.

Thus, a powerful method to index or "key" documents is required in order to ensure retrieval. The viability of the information base concept hinges on the strength of its keyed retrieval capabilities. Specific requirements are discussed in the Implementation section later in this paper. However, assuming that sufficiently powerful keyed retrieval is available, a serious question arises. How should these documents be keyed? By company name? Person name? Subject? Date? Filing by only one key provides little assurance that desired documents can always be found. For example, a file organized by company name is of little value in instances where only a company representative's name is known. One option is to maintain multiple physical copies of the documents, each filed under a different key. But the cost of maintaining multiple copies can be prohibitive, and the likelihood of inconsistencies between the files grows quickly. How does an information base solve this dilemma?

An information base frees a company from inflexible physical filing strategies. A document is physically (or electronically) filed using a unique document number (or name) that is assigned to it. The document is then electronically catalogued in an information base by those keys that provide optimum retrieval flexibility for the type of information being filed. Thus, the document number, several keys, and typically a one or two line document summary might be stored in the information base.

To retrieve a document, it is first located in the information base catalog via the key that is most appropriate. Document summaries would aid in the screening process, especially if several documents qualify based on the key and key value used. The document's unique identification number is then used to retrieve it from the physical file in which it resides.

Documents can be filed by any mix of keys desired; the mix can be varied from file to file. Company name, person name, date, subject, contract, phone number, zip code, author, publisher, product — all are examples of potential keys. No matter how many keys are used, only one physical copy of the document is needed.

Document management is an excellent information base application for several reasons:

- The majority of documents that a company has in its possession are hard copy. Little has been done to date to apply computer hardware and software solutions to the problem of hard copy document management.
- The term "document," as used throughout this paper, is a very broad one. It includes correspondence, periodicals, data sheets, brochures, books, catalogs, contracts, etc. Documents that must be managed originate both from within and without the company.
- The wealth of information contained on hard copy documents that a company customarily receives is virtually limitless. By employing information base management techniques, far more of this information can now be exploited than ever has in the past.

Although document management is huge in scope, it is only one of many potential information base applications. Some examples of other applications include:

- On-line Rolodex-type files.
- Corporate directories, including phone "books."
- Human resource or component-product information bases.

These applications would likely maintain all data on-line, unlike the sample document cataloguing system, which typically maintains the bulk of the data in physical files. Regardless of whether physical files are used or not, there is no logical distinction between the two strategies. An information base is not constrained to be totally resident on one computer system, but can be the integration of several information stores into one logical entity.

INFORMATION BASE IMPLEMENTATION

To implement an information base as conceptualized in this paper, the following conditions must be satisfied:

- Keyed sequential access must be available to entries in the information base. Keyed sequential access includes generic retrieval, ascending sequential retrieval, and if possible, descending sequential retrieval.
- The ability to key (index) information by multiple keys is required. Typically, 3-5 keys are sufficient, but an upper limit of 8 or possibly more might be required in some situations.
- A catalog approach to managing textual data is

needed to manage hard copy documents, and optionally, on-line documents. Use of this technique makes an information base invariant to whether documents are stored on or off line.

- Data security, high performance in multi-user environments, and high reliability must be ensured.
- A convenient query facility must exist for user interface to the information base.

By definition, flexible and powerful retrieval capabilities are the essential element of the information base concept. An information entry must be accessible even when the exact value of its key is not known. For example, a search for a document by the author's name must be ensured of success even if the name is not fully specified. There are several reasons why retrieval by partial key, known as "generic" retrieval, is extremely important:

- The correct, full key value may not be known — frequently the case with names of all types.
- The correct key value may be known, but the information may have been originally entered with the key value incorrectly spelled.
- Lengthy keys are time consuming to enter fully and precisely. Often, the first 4 or 5 characters of a key are sufficient to select the desired entry. For example, "HEWL" is sufficient to uniquely select "HEWLETT PACKARD" if no other company in the information base has a name beginning with those four letters.

Keyed sequential access, which includes both generic and sequential retrieval, is an absolute necessity when information is keyed by names of any type. Without it, information can easily be lost if key values are incorrectly spelled during entry. Correct spelling is a severe restriction with names, because it implies exact punctuation, use of spaces, etc., not just correct spelling of each component of a name. For example, "TRASKO,MARK S," "TRASKO,MARK S.," and "TRASKO MARK S" are worlds apart if generic access is not available. Thus, an exhaustive search of the information base is required to locate entries with misspelled keys unless the incorrect spelling can be exactly guessed. With keyed sequential access, a reasonable guess can be made (generic retrieval) to get close, in alphabetic order, to the desired entry. Then, names can be scanned in forward or backward order until the desired name is found, similar to searching for a name in a phone directory.

The need for keyed sequential access is met by IMSAM/3000, the IMAGE Sequential Access Method. IMSAM, an enhancement to Hewlett Packard IMAGE, provides keyed sequential access, including generic retrieval and ascending and descending sequential access, to entries in IMAGE data sets. An IMAGE database enhanced with IMSAM/3000 serves as an excellent facility for an information base, meeting the first four implementation criteria outlined at the beginning of this section. Entries in an IMAGE database may be keyed by up to 16 items, any number of which can be designated to have IMSAM access. Since IMSAM is totally implemented under the IMAGE umbrella, all of IMAGE's data security, performance, and reliability features are maintained. In addition, existing IMAGE application programs and tools may be used to access the information base.

The fifth requirement for an information base approach to information management is met by Datadex/3000. Datadex, a specialized query facility employing IMSAM, provides a powerful and convenient way to access information in an IMAGE database enhanced with IMSAM. Datadex provides a full set of commands that allow information to be added, deleted, modified and retrieved in several different ways.

Five commands are available to exploit IMSAM access capabilities. The Datadex Find command allows retrieval by partial key, while using any of the five relational conditions (=, >, >=, <, <=) to control the retrieval. For example:

F COMPANY = HEW finds the first company whose name starts with "HEW."

F COMPANY > HEW finds the first company whose name starts with "HEX" or higher.

F COMPANY <= IN finds the first name in descending order (the highest) starting with "IN" or less.

F COMPANY-REP < SN finds the first name in descending order (the highest) starting with "SM" or less.

Unlike Query, which allows similar operations, an entry is retrieved immediately by Datadex, because IMSAM supports keyed sequential access. The capability to do relational Find operations is thus built into the structure of the database. With Query, a Find that does not specify a key value fully or does not use the "=" relational condition requires an exhaustive serial read of the data set. This can take hours to complete on a large database.

The Datadex Next and Previous commands are often used following a Find command. Next and Previous allow the user to browse forward or backward in sequence of any key desired, examining entries one at a time. The List command allows listing of a range of entries, in ascending or descending key sequence, on the line printer. For example, all companies starting with the letters I through M could be listed by:

L COMPANY = I / M for a listing in ascending key order.

L COMPANY = M / I for a listing in descending key order.

The Xfer command works much like the List command, except that entries may be transferred to an MPE file, another database, or the terminal screen, and data is not formatted. This command allows a range of en-

tries to be transferred to a holding file or data set, then reported on using Query or a vendor supplied or user written program.

Datadex provides a turn-key facility that can be used to implement numerous information base applications. All functions required to maintain and access on-line Rolodex-type files, corporate directories, document cataloguing, and many other applications are provided by Datadex. Design an IMAGE database to fit your needs, and Datadex will do the rest.

CONCLUSION

The information base concept will play a crucial role in future office environments. It may well surpass conventional, outbound communications office automation tools in importance. Information bases provide needed information to individuals at all levels of a company. The resultant increases in productivity and efficiency, and the ability to base decisions on the best information that exists, will provide a company with a competitive edge.

Using IMAGE as a foundation, adding the power of keyed sequential access with IMSAM/3000, and employing Datadex as a query facility, an information base can be implemented with capabilities that equal or surpass any facility on any computer system available today.

Job Costing on The HP3000

Steve Perrin and Robert Lett
Bellamah Corporation
Albuquerque, New Mexico

Bellamah Corporation is one of the Southwest's leading diversified real estate developers with operations in Arizona, New Mexico, Colorado, Oklahoma, and Texas. Our construction management information requirements cover Land Development, General Contracting (light commercial), and Housing Divisions. Our Job Cost System was developed to be a management and operations tool in controlling costs and financing projects under construction. This system is a user-oriented operating system and was designed cooperatively with participating management and staff from the above divisions. The Information Services Department coordinated the development, design, and implementation of the approximately sixty stream and twenty screen COBOL Programs on our HP3000-III System. At present, three divisions and one major land joint venture are using the system across a multi-state operating environment.

The main objective of the Job Cost System is to assist divisional management in maintaining control of numerous jobs while maximizing the profitability of each job under construction. The system is capable of supporting these objectives for the following types of projects:

- General Contracting
- Housing
- Joint Ventures
- Lots
- Multiple Family Units
- Projects
- Subdivisions
- Tracts

In addition, the system is on-line oriented and automatically supports our other corporate information systems. A database has been developed to facilitate the use of advanced query and report writers such as QUIZ which is currently installed. The system should have the capability to interface with the following future expansion requirements:

- Purchase Orders (installed in housing)
- Projection Analysis on Prices, Costs, and Estimates (installed in housing)
- Percentage Completion Reporting
- Estimating
- Bill of Materials
- Scheduling
- Cash Flow Analysis
- AIA Billing Calculations
- Retention Calculations
- Customer Profiles
- Unit Cost Control

Before proceeding with a look at some of the output reports, let's take a brief look at the information flow and job cost database.

(See Page 2)

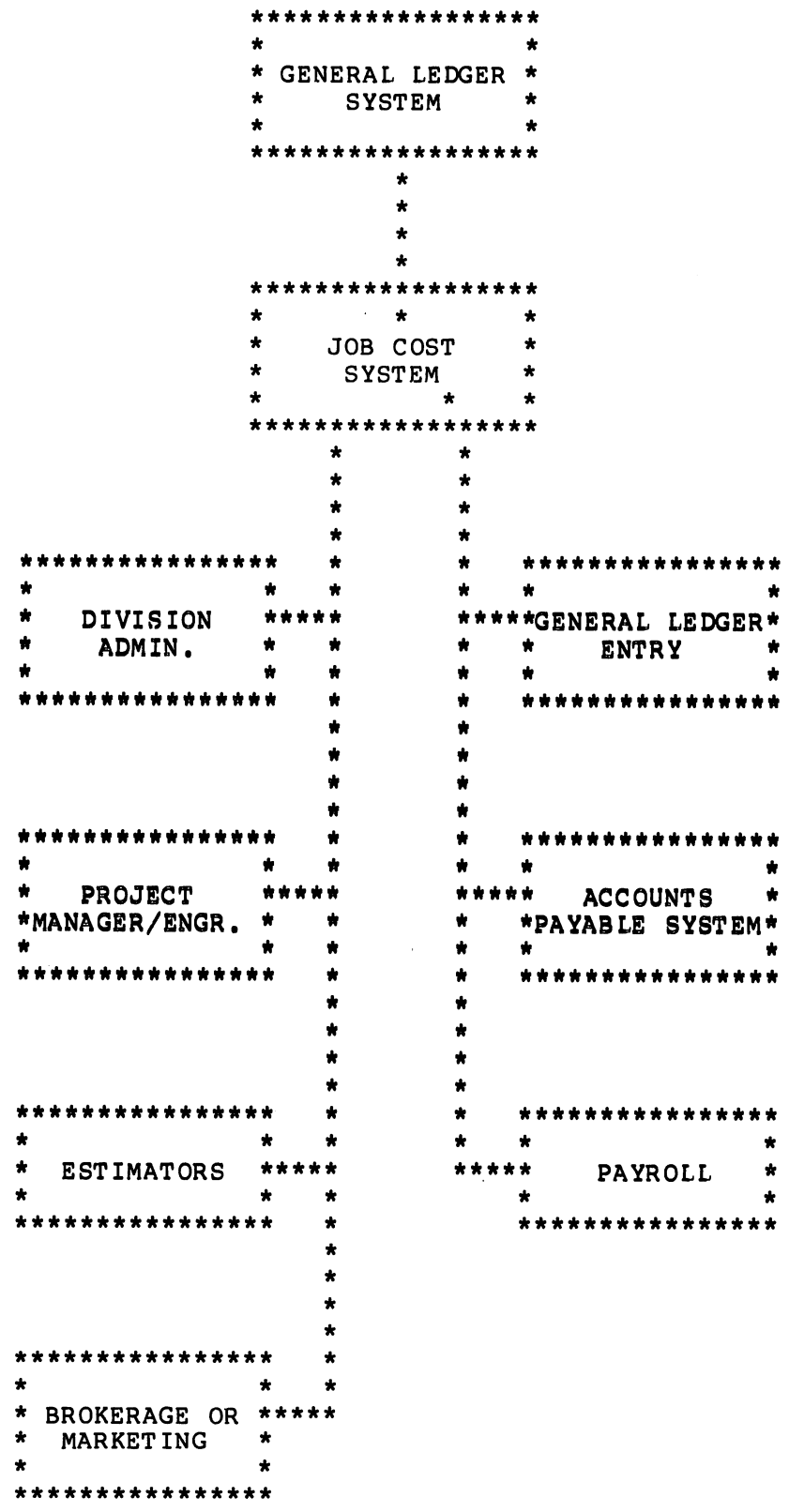
The three accounting based systems, general ledger, payroll, and accounts payable, gather financial transactions that are fed to the job cost and general ledger databases. At the present time we are processing these systems in an on-line data capture environment. As you can see from the diagram, non-financial transaction information may be entered into the system by project managers, estimators, division administrators, project engineers, brokerage and marketing personnel. The non-financial data elements break out primarily by functional area such as changes in construction status, marketing status, tax rates, zoning, estimates, sale dates, etc.

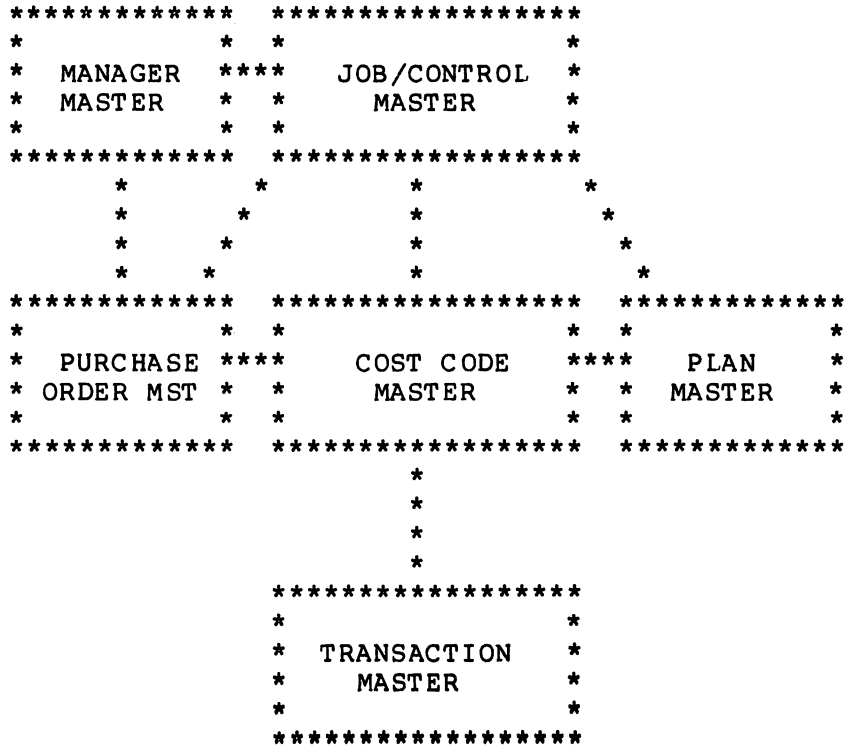
Principle data sets of the job cost database are shown in the following diagram:

(See Page 3)

The job master data contains information for each job/project/parcel and various control levels such as subdivision, city, function, and region for housing. General contracting and land differ, but are basically similar. These records are created by processing a start order or through file maintenance. The cost code master summarizes data by cost code within a job/project/parcel. These records are read from the plan master and loaded at the time start orders are processed or through file maintenance. The transaction master holds all financial transactions for each job by cost code. The purchase order data set contains all outstanding purchase orders for each job and is presently used only by the Housing Division. The manager data set is used to measure performance and summarize information by project manager. The plan master data set contains estimates and information by cost code and phase for each model elevation under housing and a standard set of cost codes with no estimates for land. These estimates are entered by file maintenancing the parcel later. Gen-

JOB COST SYSTEM
INFORMATION FLOW





eral Contracting loads their estimates by file maintenancing the cost code and estimate after the start order has been entered since each project is unique.

We will now look at some of the output reports grouped by Housing, General Contracting, and Land Development. The orientation will generally be from senior management to divisional administrative personnel.

HOUSING

Housing divisional management uses the job cost system to maximize unit profit, control costs and jobs, and finance jobs under construction. To review unit profit a Profit Analysis (Fig. 1, Appendix A) is prepared showing the gross profit amount, gross profit as a percentage of sales, profit per square foot, and net profit estimated along with column subtotals. To get an overall look at each housing job, a Job Cost Status (Fig. 2, Appendix A) can be run anytime. To examine the construction and marketing status, a Sales Analysis (Fig. 3, Appendix A) is used to see what stage construction and marketing are in. To project a sales price based on hard and lot costs, factors for gross receipts tax, profit and overhead, interest costs, closing costs, marketing commissions, discount points, lot cost, and marketing price may be entered by subdivision and a Housing Inventory Price Projection (Fig. 4, Appendix A) run by subdivision. These projected sales prices are compared to those entered by brokerage marketing showing differences with appropriate subtotals. Factors may be varied for different "what if" situations used in examining selling prices and costs. A Builders' Risk Insurance Report (Fig 5, Appendix A) is run to determine the value of all open houses under construction. This value is then submitted to our insurance company for purposes of determining insurability under our builders' risk policy.

To assist divisional management in financing housing, an Appraised Value Report (Fig. 6, Appendix A) is prepared and given to our various lending institutions. This allows us to borrow up to a negotiated percentage of the total loan value. This report shows the lending institution the status of each job and our total costs to date by each job with column totals by subdivision and city. In addition to the Appraised Value Report, a Housing Inventory Evaluation Report (Fig. 7, Appendix A) is prepared showing the lot fair market value, construction cost, total costs, market value, and sales price for each job under construction grouped under sold houses, unsold houses, and vacant lots with appropriate subtotals.

Although housing division middle management and project management have access to most of the previously described reports, they make a large number of decisions base on data at the cost code level within a job.

The Housing Work In Process Report (Fig. 8, Appendix A) summarizes charges for each cost code by major

category such as subcontract, labor, materials, and other. The cost codes are totalled and compared to a base estimate to develop a variance amount which is printed in the right hand column if it exceeds a predetermined amount. Subtotals are taken on hard costs for subcontract, labor, material, other, total costs, total estimate, and total variance. Job totals are also taken on the above plus non-hard costs. At this point variance figures are calculated for subcontract, labor, materials, other, and total costs. Various information regarding job status is printed at the top of this single page per job report. A manager may find it necessary to look at the transactions for a particular cost code. If further detail is needed, a Job Cost Ledger Report (Fig. 9, Appendix A) can be requested showing a complete transaction history by cost code by job. Cost code totals are again compared to base estimates developing variances. An option exists to select only cost codes having variances if desired. The detail transaction run is occasionally used by accounts payable personnel to verify that a certain vendor has received payment.

In addition to the above, an individual cost code may be displayed showing a summary of its charges, estimate, variance, and outstanding purchase orders. Detail transactions are displayed below the summary line followed by outstanding purchase orders.

The plan master data set is used to load master base estimates for a particular housing model and elevation. Estimators and project managers are primarily concerned with the use and upkeep of this data set. The Plan Master Listing (Fig. 10, Appendix A) shows the individual cost code amounts by subcontract, labor, material, and other along with subtotals by major category and other miscellaneous data by model and elevation. Cost codes are loaded into the cost code master either at the time a start order is processed or by entering file maintenance. The Plan Master Phase Listing (Fig. 11, Appendix A) is a finer break out showing all phases under cost codes within a particular model and elevation. We use the phase codes to print the purchase orders. In addition to hard copy, a cost code and its associated phases may be displayed on a terminal. The estimators or project managers also have the ability to compare different model cost codes by amount and dollars per square foot to see if anything looks out of line using the Plan Master Cost Code Comparison (Fig. 12, Appendix A). For a quick overall summary, they can run a Job Cost Estimate Listing (Fig. 13, Appendix A).

QUIZ can be used to display and print numerous combinations of existing data elements from the plan master, job cost master, cost code master, and purchase order master data sets. Using a generalized query/report writer greatly enchances the ultimate use of the system.

GENERAL CONTRACTING

General Contracting management requests a Work In Process Summary (Fig. 14, Appendix A) to review their overall condition. Middle management and project managers generally refer to a Work In Process Cost Code Summary (Fig. 15, Appendix A) to review charges, base estimates, variances, contract amounts, percent complete, and balances to complete figures by cost code for a particular job. If further investigation is needed, a General Contracting Ledger Report (Fig. 16, Appendix A) may be run to examine the transactions supporting each cost code. Copies of report may be requested by the owner or architect on some jobs. Individual cost codes may be displayed here as described in the Housing Section. The facilities of QUIZ are also available to the management in General Contracting.

LAND

Land Division management can request a Vacant Land Inventory (Fig. 17, Appendix A) to get an overall picture of their operations or to use with potential buyers, lending institutions, and at periodic pricing meetings. Land Division management and project engineers may obtain a Job Cost/Variance Report (Fig. 18, Appendix A) which contains a detail listing of charges by cost code with appropriate subtotals by cost code, parcel, city, and state. Summary figures are compared to base estimates which are loaded at the time start orders are entered or later file maintenanced to determine variances. Miscellaneous information is printed at the top of the page such as zoning status, number of acres/lots, etc. Individual cost codes may be displayed here as previously referred to in the Housing Section. Again, the full facilities of QUIZ are available. An Appraised Value Report by lot is available and is similar to the one described in Housing.

These are some of the ways that Bellamah uses its Job Cost System to monitor costs, control projects, and finance projects on our HP3000. We have left the Purchase Order Subsystem for another time due to the length of this presentation. It covers the area of manager performance and additional disbursement analysis.

Thank you for your interest in our area. We would be happy to answer any questions you may have.

Is a Packaged Program the Answer? A Compromise to MM3000

James G. Raschka CPIM
Key Tronic Corp,
Spokane, Washington

OVERVIEW

Many software vendors selling expensive inflexible packaged manufacturing systems lack the incentive for a pre-sale investigation to ensure success. As a result, the successful installation of manufacturing packages nationwide has been less than 10 percent. This paper will present a compromise as it relates to a real experience. It should be of interest to software vendors, manufacturing users, as well as the system designers and programmers.

Key Tronic Corporation supplies 38 percent of the world's custom terminal keyboards. The company was founded in 1969 by Mr. Lew Zirkle in Spokane, Washington. The company is privately held and has expanded to 1,200 exployees located in five locations around the city. Sales are increasing at 30 percent per year with three times the present business forecasted by 1985. During the past nine months we have been working hard to establish a better information system to handle the high volume of orders for both now and in the future. One of the main reasons for our position in the marketplace is our rapid turnaround from customer drawing to a quality finished product. Some of our customers include IBM, Wang, Xerox, Exxon, Tandy, Memorex, as well as many others. Our typical order through manufacturing cycle looks like Figure 1.

Manufacturing is very vertically integrated — meaning that practically everything on the keyboard is made from raw material. The keytops and switches are made from raw plastic pellets, printed circuit boards are cut from large sheets of laminate and etched in our own tanks, and most parts are inserted with the aid of automated equipment. All piece parts must be ready to go together at final assembly according to a predetermined schedule. Herein lies the complex data handling problem. There are over 100,000 parts and 350,000 structure relationships that must be coordinated with the 16 week backlog of piece part and keyboard orders.

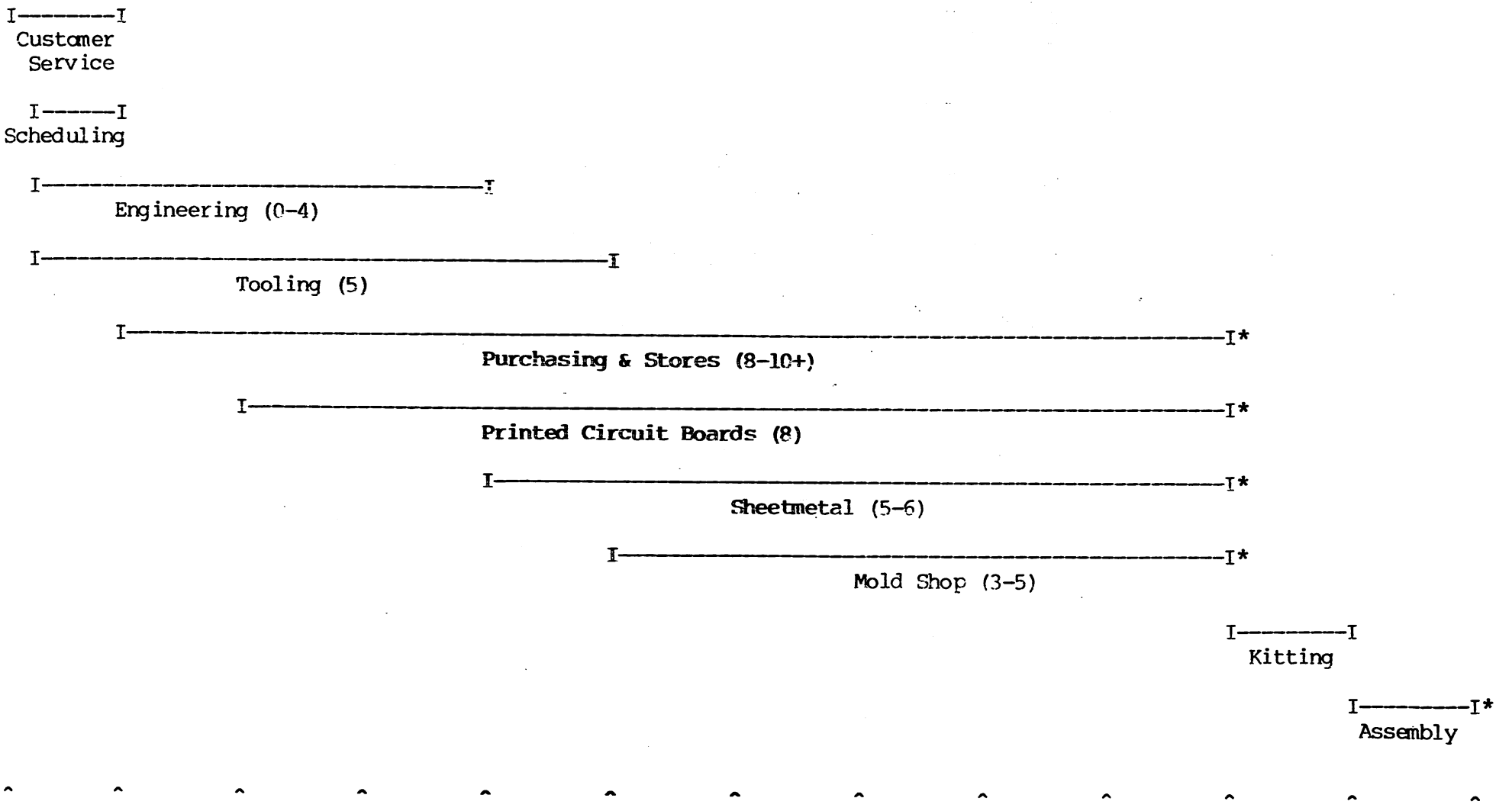
The use of computer systems to aid in the tracking of information for the company has evolved through a number of minicomputers and stand-alone word processing systems. Until a few years ago a central computer system tracked mainly accounting and some of the 2,500 electronic parts requirements. Small systems such as a Burroughs, two IBM/32's, and an IBM/34 were used. For three years we searched for a packaged manufacturing system to meet our growing needs. In November of 1980, a Hewlett-Packard 3000/III computer was installed with the Materials Management/3000 software developed by HP. We will trace the initial failures, eventual successes, and present status of this installation. The final step was to scrap the programs from MM/3000 and write our own to the database that had been created.

In conclusion we will discuss how vendors might better sell packages, especially in the light of past failures. We will discuss preliminary system study, programming needs, educational needs (outside of HP), and follow-on consulting. This will be a constructive presentation and should help future HP3000 manufacturing systems to be brought up successfully.

DETAIL

We tried to start out right. One of the first things the MRP gurus tell you is the need for education from top management on down. The only training class that is offered with this package is System Administration. The S.E.'s will tell you that this course is for one person in the company, the one who will manage the database, programs, and train the users. We sent the Director of Engineering, Manager of M.I.S., an Mfg. Project Leader, a Systems Manager and a Programmer. As time went by only the Director is still involved with the program. Our present MM/3000 system manager has never taken the class. Looking back the money would have been better spent sending the line supervisors to a generalized course such as Oliver Wight's five day MRP class or some of the local American Production & Inventory Control Society (APICS) training classes.

The next misjudgement made was in the estimate of the database size and the amount of hardware, especially disk space that would be needed. Coupled with the fact that no one was yet using MM/3000 and the designers never planned to have it access such a large database, we had a great deal of difficulty trying to make a successful MRP run. The present database takes one and one-half 120 megabyte disc drives. As an example of miscalculation, it took us five days to load the



12 WEEK MANUFACTURING CYCLE
KEYBOARD MANUFACTURING

NOTE: *Quality Control

Figure 1

database running twenty-four hours a day. As soon as the MRP generation starts, it needs 1,000,000 bytes of free space. It is efficient in the sense that it cleans up files as it goes. We often had more free space after a run than before. The system also keeps track of the size of temporary files from run to run. A typical run takes 14 hours even if we have only ten master scheduled items.

As time rolled on it became apparent that factory help was needed. For nearly three months we were seldom successful at making the system run properly. We had little help from PICS because nobody had a lot of experience since MM/3000 was new. After a while we had direct access to the writers of MM/3000. It should be noted here that the vertical rather than horizontal integration of the HP local and regional offices made it difficult to get all the right people together to resolve the problems. Some of the surprises we found were that you can only have 64,000 parts per key. It took two days to break this down to smaller sets, after we crashed. The system did not have the capability to copy one bill of material to another. We finally wrote our own routine to do it after HP had tried for eight months. MM/3000 and in MPE IV eat up 80-90% of the CST's. Even the addition of MPE IV does not show any improvement. As far as we have been told the HP3000/64 computer upgrade won't help initially here either as it will have the same number of CST's in the initial operating system.

With all these start up problems we still feel we are on the road to success. As the president of the company put it, however, had it not been for the recession that moderated our annual growth to only 30%, we would never have been able to keep up with the business. Now we should begin to look at datelines to see what events were set in motion to help build a useful information database.

4/1/81: With the help of a local HP/3000 with a card reader the keypunched card structure database was put on tape. During the same period packages were bought for the accounting department. Database conversions were made from IBM eight inch floppies. This included the G/L, A/R, A/P, Payroll, and fixed asset systems.

4/20/81: Layed out a three year MIS plan and hired a manager to make it go.

5/1/81: The Key Tronic MRP system for electronic parts was rewritten and completed on the HP/3000 (MRP/KTC). We didn't want to change this until we were satisfied that MRP/MM3000 could handle it. One week after he first converted it, the programmer left us and went to work for HP.

5/1/81: Turned the IBM system off.

5/15/81: Sold and shipped the IBM/34.

6/15/81: Started our first month-end accounting close.

7/1/81: Started our first year-end accounting close. During the past two months it became obvious that with 32 users the response time was going to heck in a hand basket. HP has some performance charts that show you what happens.

8/1/81: Installed our second HP3000 system and split out accounting and manufacturing.

9/15/81: Hired a senior programmer to help write a better MRP system (MRP/NEW). He did reduce the 14 hour run to less than an hour plus he fixed some problems HP had not been able to solve.

11/5/81: Installed an HP3000/33 for development work.

Since then we have continued to rewrite the software. Here are some of the items we are redoing.

- Because of the length of time it takes to get a report out we made our own MRP explosion module.
- Our company was more familiar with a "bucketed" MRP report rather than a "bucketless," so we made it bucketed.
- The structure and parts file editors locked entire data sets rather than items. This made the data entry operators very frustrated because they were forced to re-enter a data item over and over. Therefore, we went to a two step approach. The first was to use the MM/3000 batch data entry capability. We then rewrote all the editors since we did not have source code. Used PROTOS and the VIEW screens that had been established. Also used a "Father-Son" approach to programs so that the "Son" program worked with the database and the "Father" worked with the user.
- By efficiently writing our own code and taking the MM/3000 programs off, we reduced the CST's being used and, therefore, opened up the machine to more users. In two years there will be 60 to 70 terminals on three HP3000's.

CONCLUSION

No matter what manufacturing system is used, packaged or self-written, it takes up to two years to get MRP working. A packaged system may help prepare the database but it often is too generalized to meet specific company needs. The best thing for a vendor to do is to sell a skeleton system that will let the user easily build his own custom package. It would be better to spend $5,000 for the skeleton and $20,000 for six months of consulting to educate and write the finished software. HP charged us $25,000 for the package and an additional $11,000 for consulting.

In my experience with IBM, DEC and HP there usually are similar areas that sales people fall short when proposing their equipment.

1. The initial hardwre cannot handle the database.
2. The software is too generalized and cannot be customized.
3. The combination of hardware and software does not meet the response time expectations of the user.

Hewlett-Packard hardware was our choice because it could be expanded as the company grows without rewriting software. IBM and DEC usually fall short here. The software is now our own so it can be customized. Response time got pretty bad, but it looks like our adding CPUs and rewriting more efficient software will get us over this last problem. We also feel that the HP3000/64 will be our next computer upgrade, especially if a better operating system (but upward compatible to MPE IV) is developed.

All in all it has been a struggle, but we feel we are seeing the light at the end of the tunnel. For a company whose people had seen very little real time database use of a computer we have come a long way since May 1, 1981. We have been called by a number of companies for assistance and find there are many out there with similar problems. We look forward to working with the HP/IUG Manufacturing Interest Group in the future and hope that many of you will do the same.

Management Reporting With Hewlett-Packard's Decision Support Graphics

William M. Crow
Director, Systems Development
Austin Information Systems
A Division of The Austin Company

Hewlett Packard's Decision Support Graphics System for the HP3000 family computers (DSG/3000) provides an effective tool for preparing line, bar, and pie charts to graphically represent numerical data. Austin Information Systems has implemented DSG/3000 for internal reporting to the Corporate Management of The Austin Company. This is the first of a three phase program to provide graphic reporting as a component of user application systems. This paper details the results of this project.

THE AUSTIN COMPANY

The Austin Company is an international Design, Engineering and Construction Company headquartered in Cleveland, Ohio with offices throughout North America, South America, Europe and Australia. The Company specializes in all types of industrial and commercial construction, providing the owner a single point of contact for all phases of the project from feasibility study through occupancy. This "Austin Method" of integrating design, engineering, and construction provides the client very rapid turnaround, allowing earlier occupancy and providing a faster return on investment.

Throughout its one hundred year history, The Austin Company has delivered 90% of all projects on time and within budget and has built a solid reputation on innovation, reliability, and quality. The Company's motto of "Results, Not Excuses" speaks for itself.

AUSTIN INFORMATION SYSTEMS

To continue to provide its clients the best possible service using state-of-the-art technology, The Company created Austin Information Systems (AIS) in 1978. AIS is chartered with providing computer based applications for The Austin Company. This charter includes Information Systems, Office Systems, and Engineering Systems. The current AIS network of 10 computers and over two hundred terminals serves 12 domestic offices of The Company and over 20 construction field locations. The primary Information Systems are online, interactive applications for Project Management: Cost Estimating, Cost Control, and Scheduling.

A computer assisted approach to project management applications dramatically enhances the capability of The Company. Multiple alternatives can be evaluated in the cost estimate. Major changes in the preliminary design can easily be priced and evaluated in much less time than was required by manual techniques. The estimate typically provides greater detail than those prepared by hand because of the ability of the computer to comfortably process large volumes of data. The Cost Control System provides a significant improvement in turnaround for the preparation of periodic job condition statements. Cost data is continually collected at the project site and posted to the database and, combined with the Scheduling System, project trackability is noticably enhanced. This provides the client improved visibility and allows The Company to respond faster to potential cost or schedule problems.

THE REQUIREMENT FOR COMPUTER GRAPHICS REPORTING

Now that these major applications have been in place in most offices for over two years, AIS is actively developing the next evolutionary enhancements. The implementation of graphics to further improve the quality of information presented to the client is considered as one of the next logical steps.

Since graphics hardware is already in place in some offices to meet requirements for engineering applications, this will ease part of the cost burden for implementing the first project management applications.

The ideal operational mode for these applications would allow the user to generate a graph or chart of a predefined format as easily as a numeric report of a predefined format is produced. The majority of the control provided to the user is for the selection of the subset of data to be included in the chart. The specific format, axis conventions, legends, titles, colors, patterns and other specific formatting variables that are part of a typical chart would be predefined in much the same way that report titles, column headings, numerical format and column positions are predefined for numerical reports. This allows the user to concentrate on the data and not be burdened with the intricacies of computer graphics.

INITIAL ALTERNATIVES FOR GRAPHICS REPORTING

While graphics have been used for several one-time requirements in the past, the systems did not lend themselves well to operation in a "production mode" under user control. This would be required to satisfy the design objectives for integration with the interactive Estimating and Cost Control System (ECCO).

An interactive chart preparation system (MULTIPLOT) is provided by Hewlett Packard for standalone operation on the intelligent graphics terminal in use at Austin. While this system was able to generate the desired charts, its mode of operation required the user to prepare the specific data to be graphed and respond to several questions defining chart format. While usable for one-time applications by an operator familiar with the system, it did not provide any reasonable interface to ECCO for the end user.

The development of custom routines to generate the required charts could be designed to provide the necessary user interfaces but the cost involved in this custom programming combined with the inflexiblity of the final product did not make this alternative very attractive.

HEWLETT PACKARD'S DECISION SUPPORT GRAPHICS SYSTEM

HP's announcement of DSG/3000 in late 1980 provided a reasonable alternative to meet our design objectives for implementing production graphics with the ECCO System. Additionally, it appeared to be capable of addressing the volume of one-time graphs currently being prepared for other requirements.

DSG/3000 provides the capability to prepare several variations of line charts, bar charts, and pie charts. The chart specification is prepared through a friendly interactive system utilizing "fill-in-the-blank" formatted screens.

Data is extracted from sequential files that can be easily prepared from a database using a report writer. Once a chart is defined, it can be used again and again with different sets of data without the need to redefine the chart parameters. Scaling can be fixed by the user or automatically scaled by the system to fit the data. Output can be displayed on a graphic CRT terminal or routed to any of several HP hard copy devices: pen plotters, thermal plotters or dot matrix printer/plotters.

DSG/3000 provides the user with capabilities to select subsets of the data file to be included on the graph. This allows extreme points to be deleted easily or several different charts to be prepared from a single data file. The user first assigns variable names to the fields of the data file (either fixed or free format). Additional variables can then be defined by expressions using previously defined variables. Finally, conditioned expressions that control which data is actually used can be defined using any of these variables.

For one-time applications, DSG/3000 provides a formatted screen that allows the user to enter data interactively. This data can be edited and saved for later use.

The documentation provided with DSG/3000 is very good. A comprehensive User Reference Manual provides the detailed description of the system while a brief User's Guide answers most questions for the user while working at the terminal. An optional self-paced training package leads a user with no previous computer experience through the full capabilities of the DSG/3000 interactive system. This course typically requires about ten hours of reading and terminal time.

DSG/3000 provides the fundamental capabilities to meet the defined requirements because of the distinct separation of data, chart, and output definition inherent in its architecture. The user may independently define the data structure, the parameters which control the type and format of the chart, and the output scaling and destination. Any of these can be updated for the specific requirements independent of the others. This allows the system analyst to develop the required chart format definitions and data interface appropriate for the input file. The user must only define the specific data to be included in the data file and the output destination for the chart.

DSG/3000 can be integrated directly with the ECCO System (or any other user application) because all of the functions of DSG/3000 that are available through interactive, formatted screens are also available as program callable intrinsics. This provides the system designer all the capabilities of DSG/3000 while also allowing the user to be completely insulated from all of the graphics controls. This allows graphics to be implemented in a truly production mode.

A PHASED IMPLEMENTATION PLAN

While DSG/3000 appeared on the surface to provide all the mechanical tools necessary to meet the design objective, several questions regarding types of graphs, data to be graphed, output mechanism, hardware configuration, and user reaction had to be answered for our specific application, environment, and user community. To accomplish this, a three phase plan was defined leading up to the implementation of DSG/3000 with the ECCO System:

Phase I — DSG/3000 to be implemented in an interactive mode to prepare charts for AIS management reporting to Austin Corporate Staff. Data to be charted includes system utilization, revenue and operating costs.

Phase II — DSG/3000 to be implemented in a program callable mode as part of the AIS computer services chargeback system. This will provide clients graphic reports of computer services utilization, distribution, and costs.

Phase III — DSG/3000 to be implemented as a program callable component of the ECCO System as described earlier.

AIS has been using DSG/3000 for several months as

part of our Phase I program. Currently it is being implemented into the next major release of the chargeback system and following that release, DSG/3000 will be incoporated with the ECCO System.

HARDWARE REQUIREMENTS

A surprisingly minimal hardware configuration is required to implement graphics in a production environment. Since the user is not designing charts, but only generating charts based on predefined chart definitions, there is little requirement for a graphics display terminal to preview the output. The application program is operated through standard block mode alphanumeric terminals and the graphic output is routed to a four pen plotter. (HP now sells their plotters with eight pens instead of four but none of these newer models are currently installed at Austin.) The plotter is equipped with a roll paper, automatic chart advance option that allows the device to be operated while unattended. In actual production use, the plotter will require little more attention than a spooled printer serving users with printed output. The HP model 7220S plotter used at The Austin Company costs approximately $7250.00.

While not required for all users, a graphics display terminal can be made available in each office to facilitate the use of DSG/3000 as an interactive tool as well as allow charts to be previewed before plotting when required. At Austin, we are currently using HP 2647A Intelligent Graphics Display Terminals. Configured with the required options and interfaces, the HP 2647A costs just under $10,000.00. The primary use for these devices is currently for engineering applications; several unique requirements justify the cost of this sophisticated terminal. Hewlett-Packard recently announced that their 2623A Graphics Terminal provides the ideal capability set for the Austin environment. At $3750.00 this terminal provides extensive graphics capabilities for the same price of an alphanumeric terminal of 2 years ago. Also available is an integral thermal printer option for $1210.00.

AIS is currently using an HP 7310A thermal printer/plotter to produce fast, black & white graphics output for both preview and reporting. While this particular product has been discontinued by HP, there are other devices within the product line that provide similar capabilities.

The chart in figure 1a was produced on the 7220S pen plotter (the original used 5 different colors) while figure 1b shows the same chart displayed on the 7310A thermal printer/plotter.

PHASE I RESULTS

Because of its use on an interactive basis during the first phase, DSG/3000 has not truly been implemented in a production mode. While many standarized charts are produced monthly, some charts are still produced on an as-required basis. However, the exercise has provided insight to many of the issues that must be addressed to implement graphics as a component of an online, user-based application system. The charts produced required minimal effort once the initial definitions were established and have been very well received by the target audience: Corporate Staff members. The operation of an Information Systems Division within a construction company introduces unique management problems and graphics has provided better insight at the Corporate level. The enhanced clairty of the data yields measurable improvements in the ability to effectively report on the operations of the division. Graphic reports have also been instrumental in a current AIS project re-evaluating and redefining the entire computer services cost chargeback procedure.

The project has uncovered several avenues by which graphics can overburden or obscure the decision making process. Like any other tool, graphic reporting must be used intelligently to be effective. The user must understand the capabilities and limitations of the media to secure any reasonable advantages.

HOW, WHAT, WHO, WHEN

Significant in the effective use of any tool is an understanding of how to use it, what it should be used on, who it should be used for, and when it is appropriate to be used. Computer Graphics offers no exceptions.

Virtually any type of data will lend itself to attractive graphic representation. Common sense provides the best guideline to choosing the type of chart to use. Data plotted against time can best be represented with a line or bar chart. Stacked bar segments can show an additional dimension of distribution within each period. Multiple dependent variables can be represented with multiple lines plotted on a common axis or with clusters of bars at each discrete period defined by the independent variable. The former allows comparisons of trends while the latter allows comparison within each period. Care must be exercised in using line graphs with a discrete independent variable because the chart will tend to show trends that imply the data is continuous. Pie charts show percentage distribution of a single data element. The pie chart can be misused if it is not appropriate to describe the total data set as the sum of the discrete data elements represented.

While certain classes of data may lend themselves to an attractive graphic portrayal, it may not be appropriate to graph if it provides no new insight to the data. This is best described by considering

> *Crow's First Maxim of Graphic Reporting:*
> "Graphing certain classes of data is like teaching a gorilla how to speak . . . It can be done, but will it tell you anything you didn't already know?"

A graph that offers no new information can only add confusion to a decision making process.

The use of colors and fill patterns can be used to dramatically enhance the asthetic appeal of a graph but

again, common sense must dictate their use. If the graph will be reproduced for distribution, it must not depend on colors to convey vital information. While different line patterns and fill textures can be used to differentiate data, an excessive use may generate more confusion than clarity. One must maintain simplicity in the data to use colors and textures effectively in a production mode. Too much information portrayed on one graph soon becomes meaningless. The age old T-SHIRT rule of presentations holds true:

> "If it's too much information to fit on the front of a T-SHIRT, it's too much information to put on a presentation graph."

The goal of graphics is to lend clarity and insight into otherwise confusing or involved data. An effective graph will make its point at first glance and not require the user to study it in detail.

The choice of chart type, colors, textures, and data scaling can be powerful tools in controlling the desired reaction to the data. Charts can exagerate, emphasize, diminish, or obscure the information but they will not change the basic facts presented. While it may be desirable to use these tools to create an intended response, sometimes the attempt only yields confusion. In a production mode, where each graph is not given individual design consideration, the choice of these display parameters must be made carefully to be effective. The best chart is one that relys on the data for the message, not the surrounding clutter of legends, colors, fill patterns and titles.

Knowing the intended audience for graphic reporting is essential in preparing an effective chart. A skilled manager familiar with the data that is to be presented can digest far more information in a single chart than can a client to whom one is presenting new and complex information for the first time. While computer graphics can be very impressive, if they present confusion then the over all result will be negative. In some environments, graphic reports may not be appropriate at all. A senior executive that is skilled in interpreting periodic numeric reports may not gain any significant benefit from graphs of the same data. This individual's management style and mode of operation is well defined and most likely very effective. Unless there is a recognized difficulty in digesting data in a numeric form, a graphic report may not be appropriate. This situation is typically much more pronounced if the user is to use a system interactively to produce graphics on an ad-hoc basis. The best target users for this type of tool are young executives or middle managers who have not developed a complete management style and are receptive to tools that can improve their performance. A senior executive has allready developed an effective management style using the tools currently available. Unless there is difficulty with the present management approach, the senior executive is not likely to adopt new tools or techniques.

The decision of when to use graphic reports need only address the cost effectiveness of the intended application if the afforementioned issues have been analyzed properly. The cost of preparing the graphs, including one time costs and operating expenses, must be weighed against the value of the graphic reports. This analysis is no different than assessing the value of any other computer based application. If it provides a direct replacement for a task previously done by manual techniques then it can most likely be expressed in a firm cost comparison. If the graphics reports provide a new resource to the organization, the value of this resource must be assessed objectively to be compared against the cost of implementation.

DSB/3000 LIMITATIONS

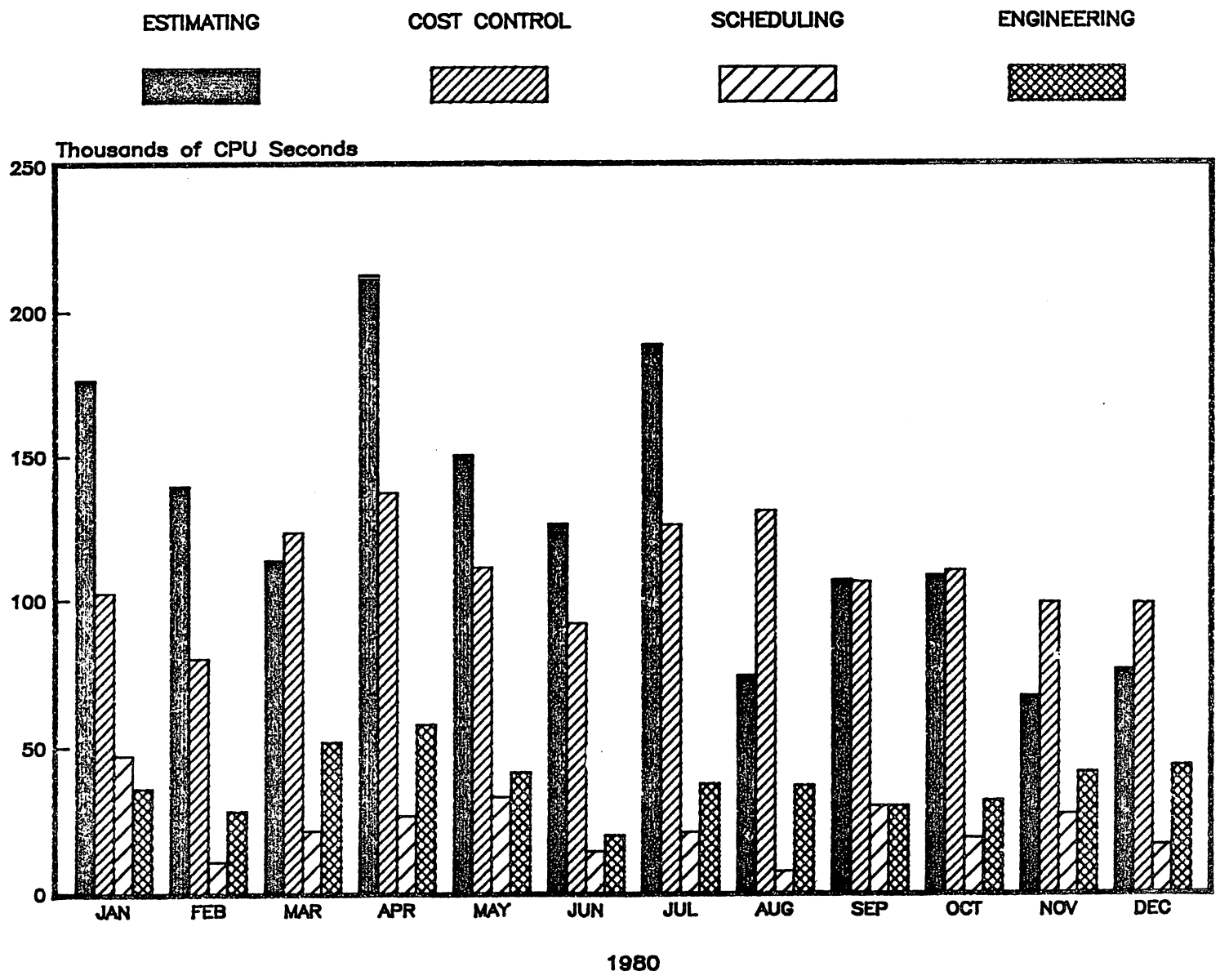
While DSG/3000 provides the fundamental tools required, it has limitations which reduce its potential effectiveness. As a system designed to be used either as an interactive application for ad-hoc reports or a production system for presentation graphs, it has compromised at both extremes by positioning in the middle ground. The volume of forms that must be completed to produce a single chart is confusing for the first time or casual user. The effort of preparing the graph soon eclipses the usefullness of the final product. DSG/3000 lacks sophisticated capabilities for easily preparing mulitple charts on a single page, overlaying line and bar charts, or utilizing a variety of character fonts; these are vital functions for preparing formal presentation graphics. The data selection and qualification capability of DSG/3000 is very useful, but too limited. The user must typically extract specific data for each graph. It would be very useful to provide DSG/3000 a direct interface to the HP/3000 IMAGE database system and eliminate the need of a report writer to extract the data to a sequential file.

SUMMARY

DSG/3000 has opened a significant door in allowing a straightforward implementation of graphics reporting a successful first step toward implementing this system as a powerful enhancement to existing online, interactive applications.

AIS Computer Usage Summary — 1980

Monthly Usage by Application — All Users



Prepared by: Austin Information Systems

7/81

Figure 1a

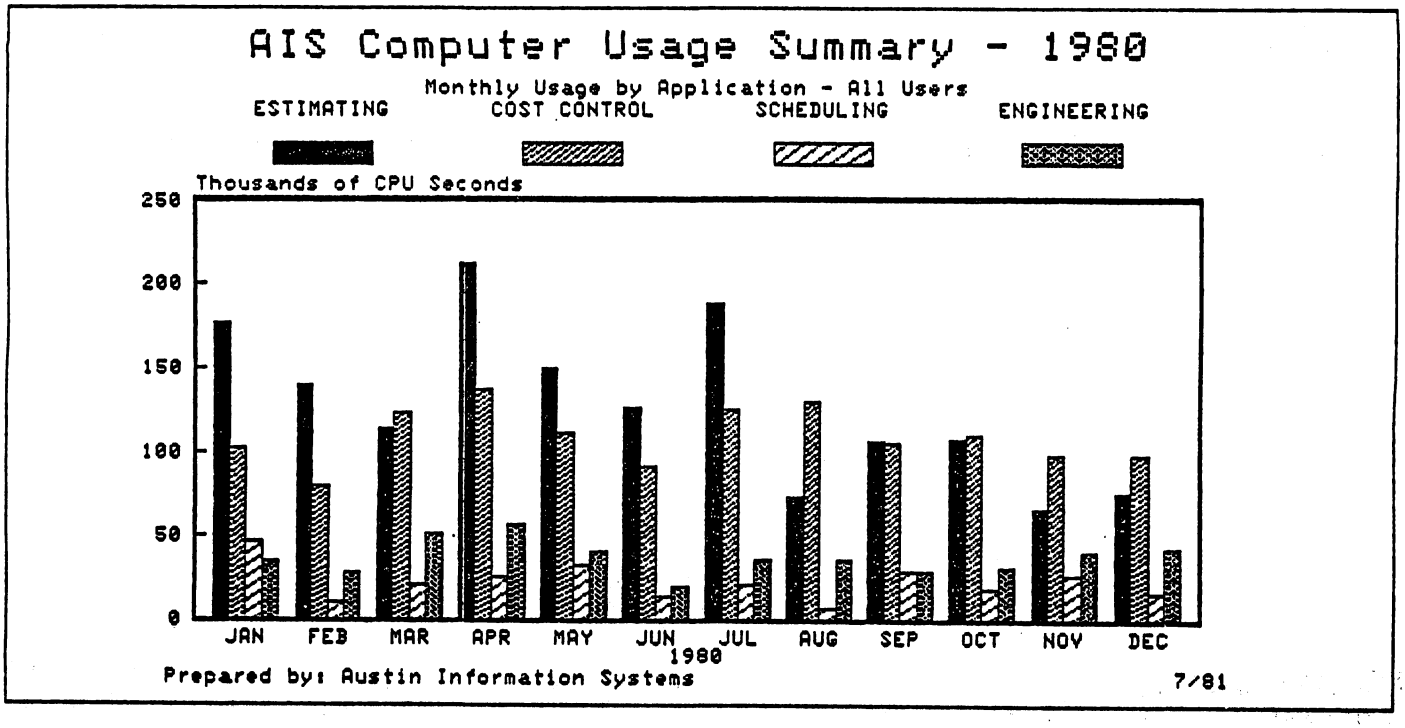


Figure 1b

TOTAL	ESTIMATE	6	126020
TOTAL	ESTIMATE	7	188450
TOTAL	ESTIMATE	8	73933
TOTAL	ESTIMATE	9	106439
TOTAL	ESTIMATE	10	107962
TOTAL	ESTIMATE	11	67009
TOTAL	ESTIMATE	12	76091
TOTAL	ESTIMATE	0	12628020
TOTAL	COSTCNTL	1	102265
TOTAL	COSTCNTL	2	79981
TOTAL	COSTCNTL	3	123160
TOTAL	COSTCNTL	4	137017
TOTAL	COSTCNTL	5	111038
TOTAL	COSTCNTL	6	91635
TOTAL	COSTCNTL	7	125674
TOTAL	COSTCNTL	8	130609
TOTAL	COSTCNTL	9	105769
TOTAL	COSTCNTL	10	109804

A Portion of data file DATAPLT3

Figure 2

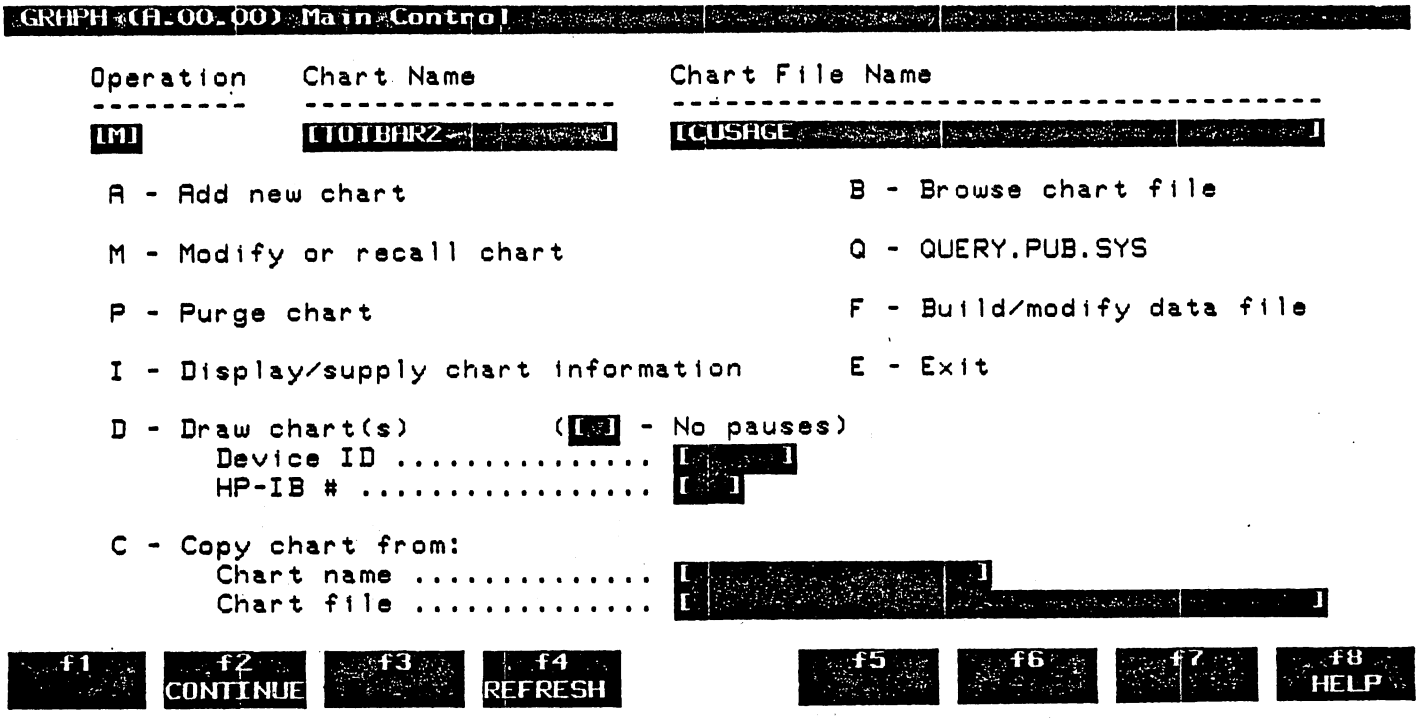


Figure 3a

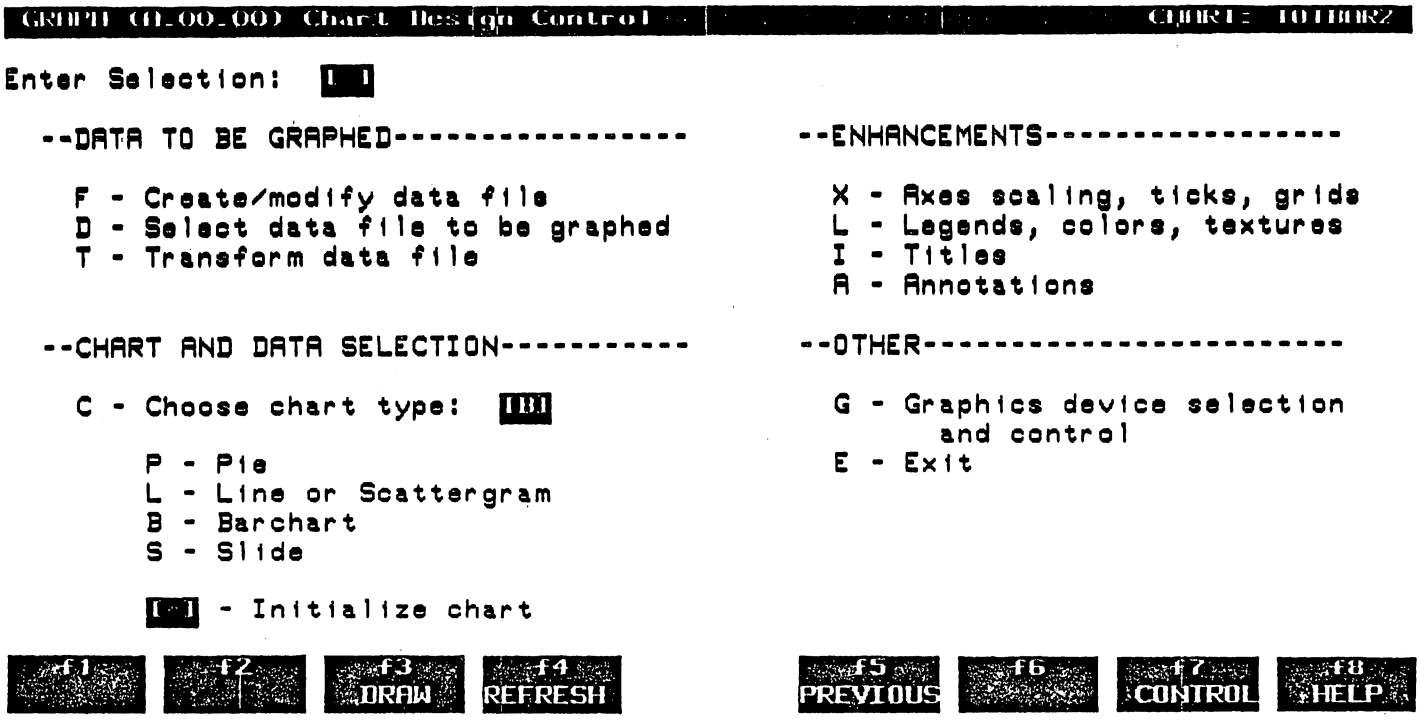


Figure 3b

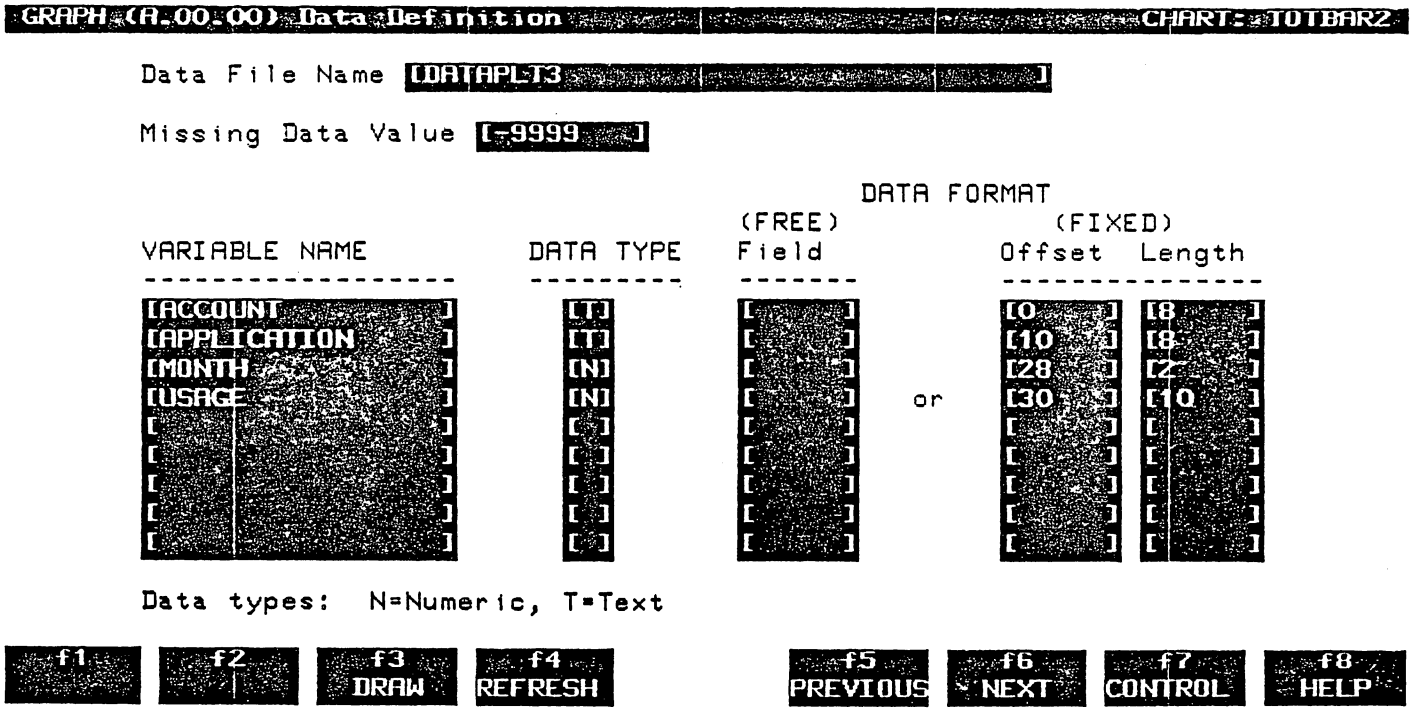


Figure 3c

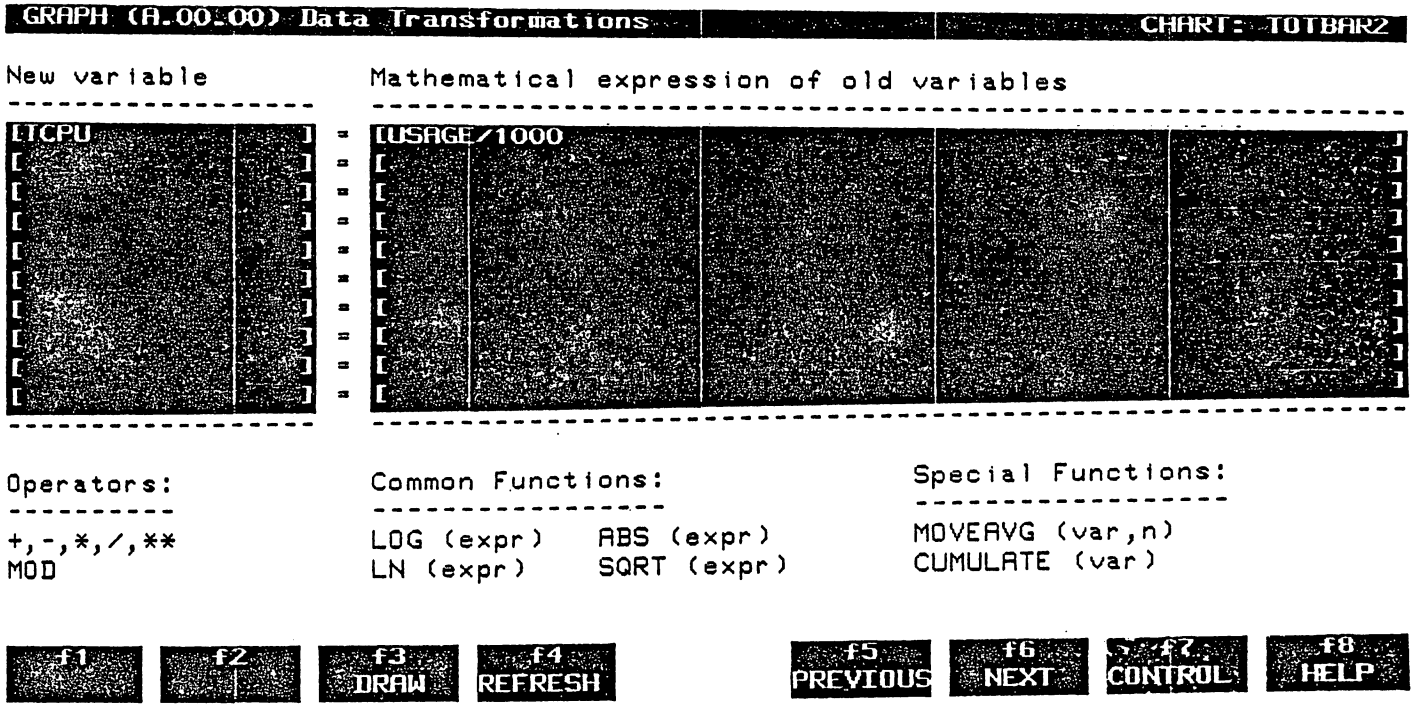


Figure 3d

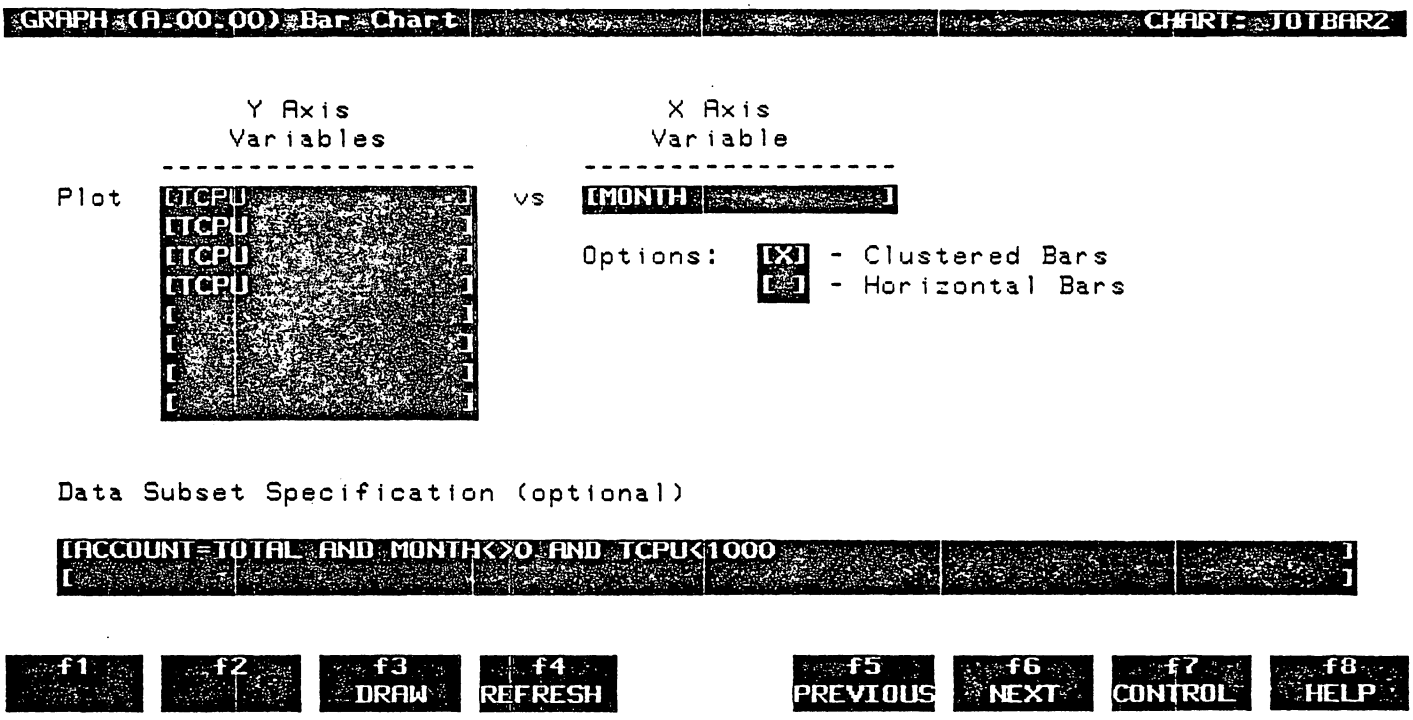


Figure 3e

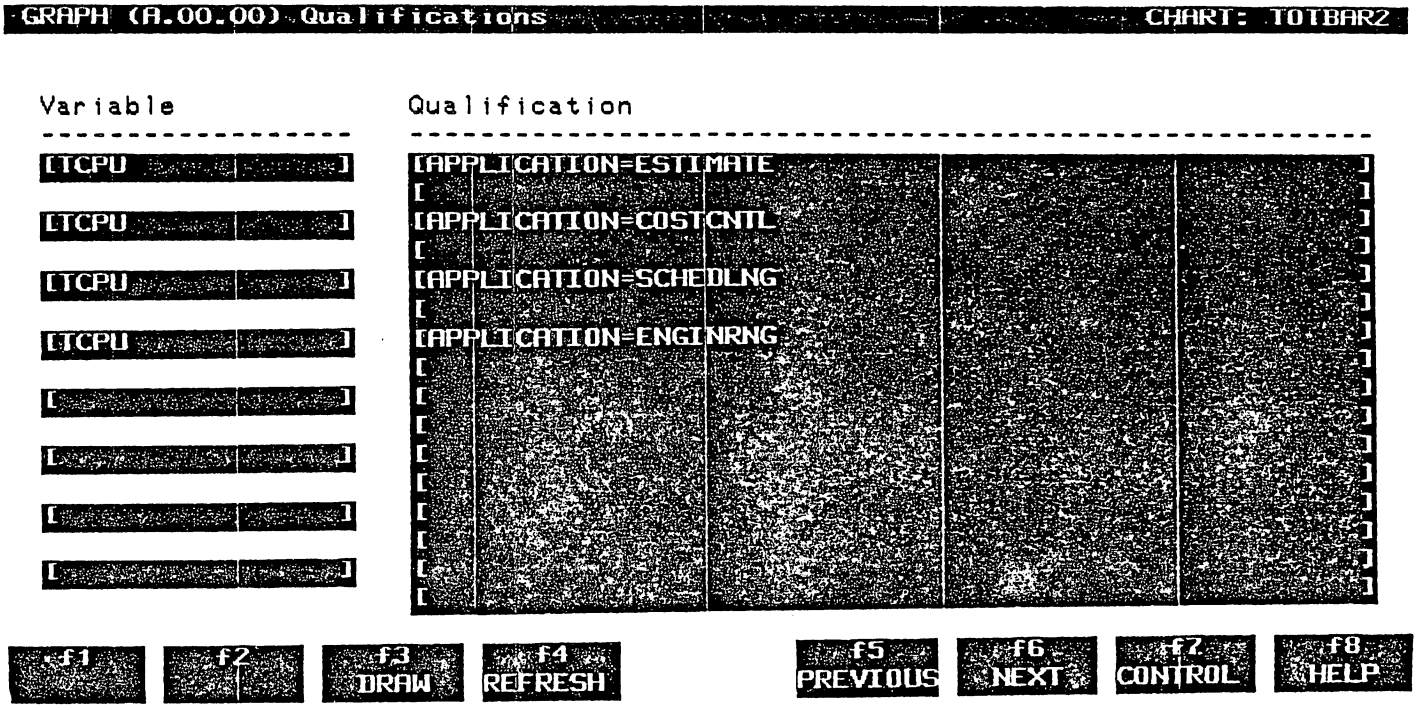


Figure 3f

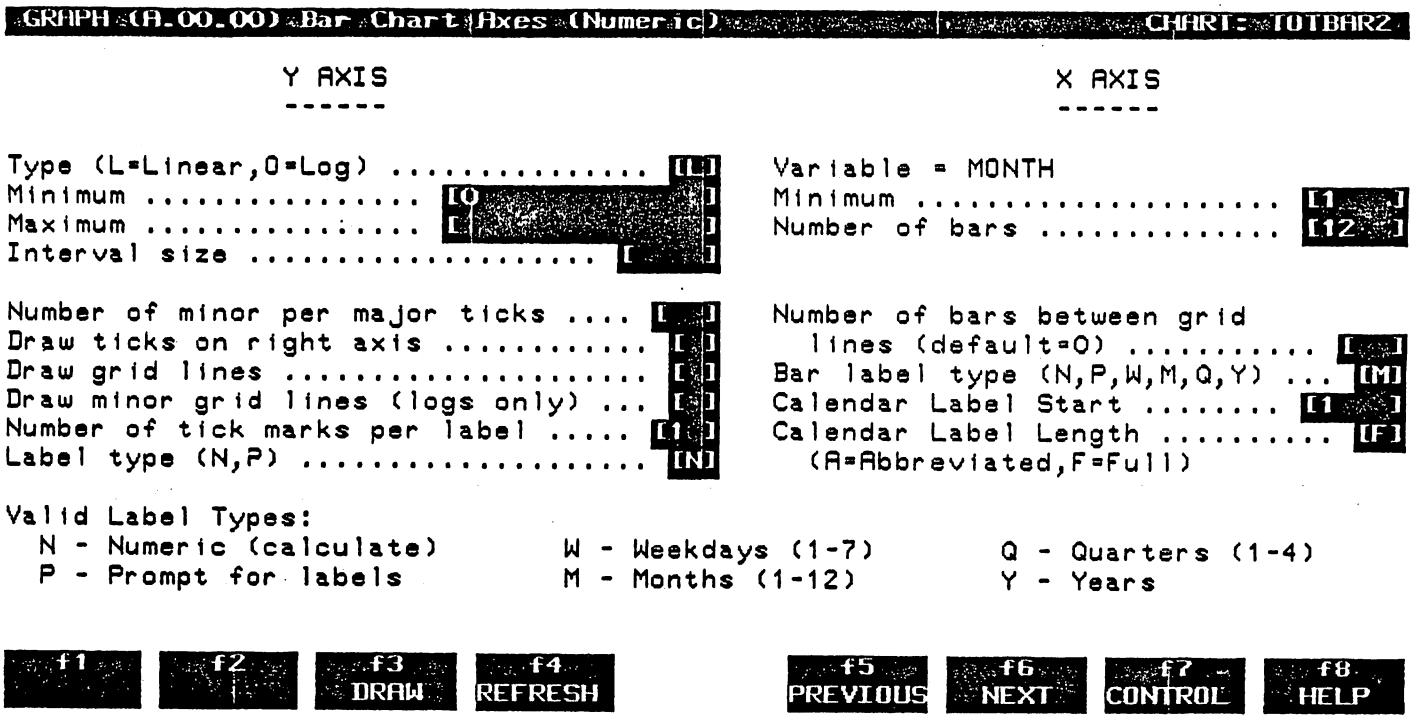


Figure 3g

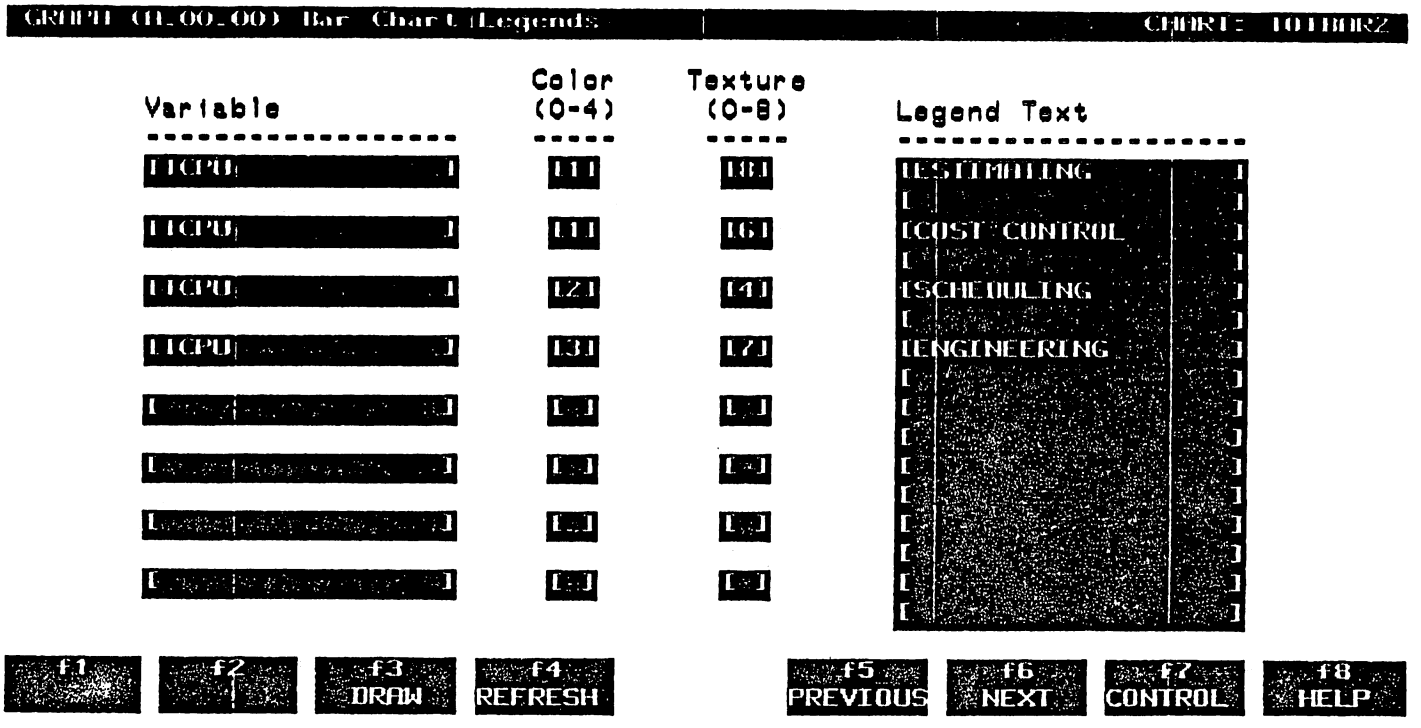


Figure 3h

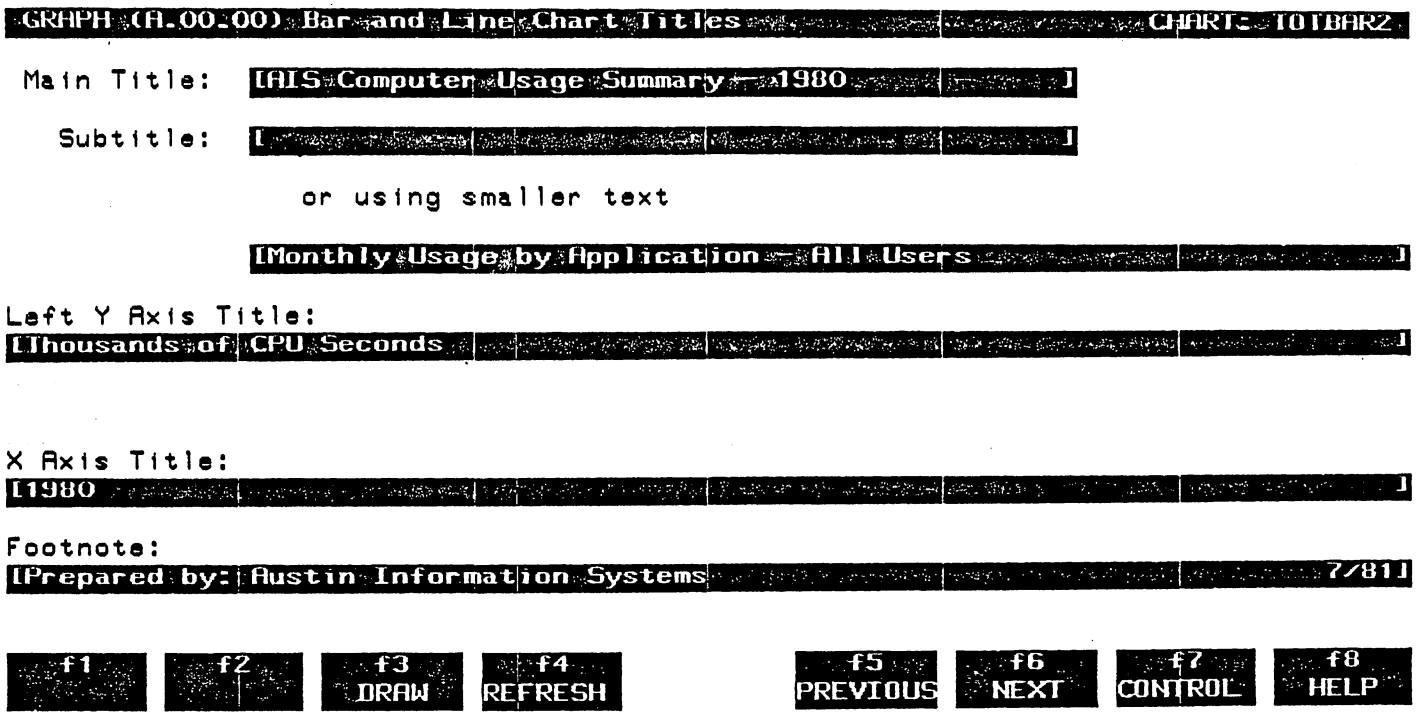


Figure 3i

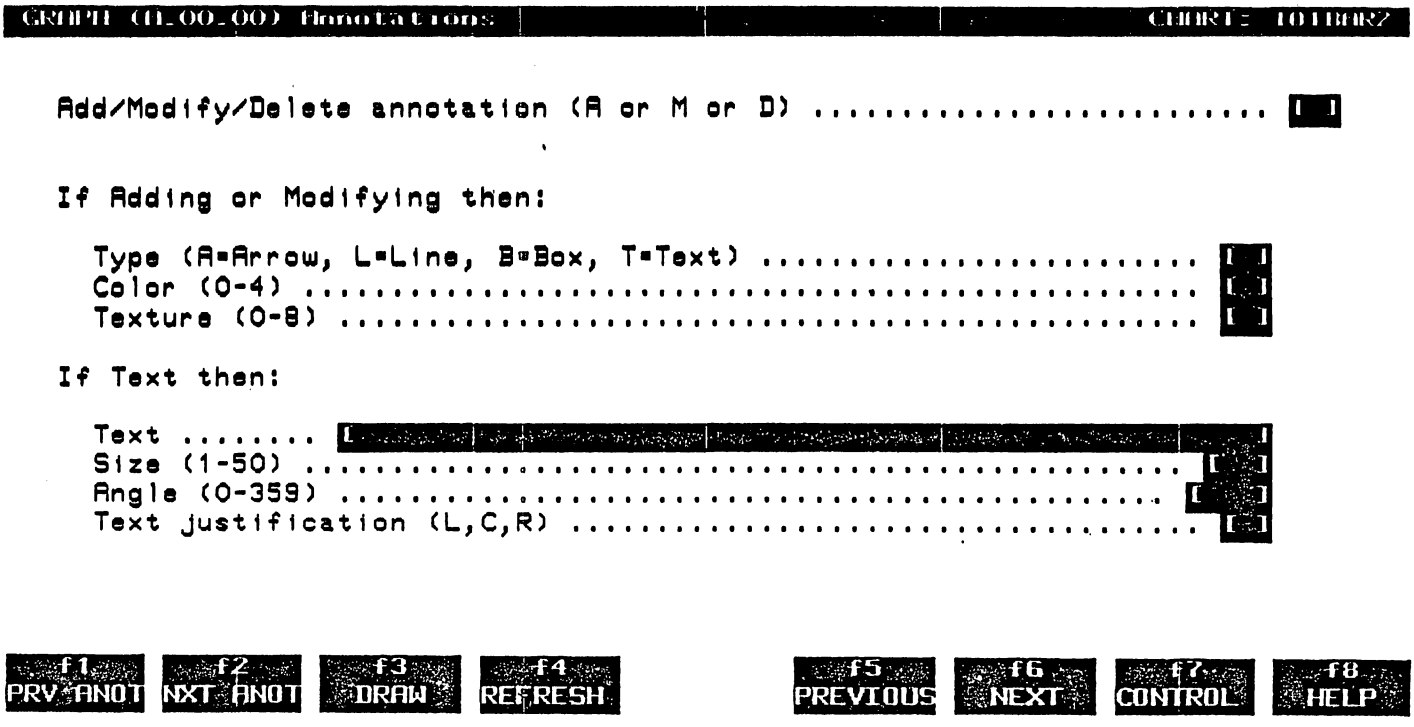


Figure 3j

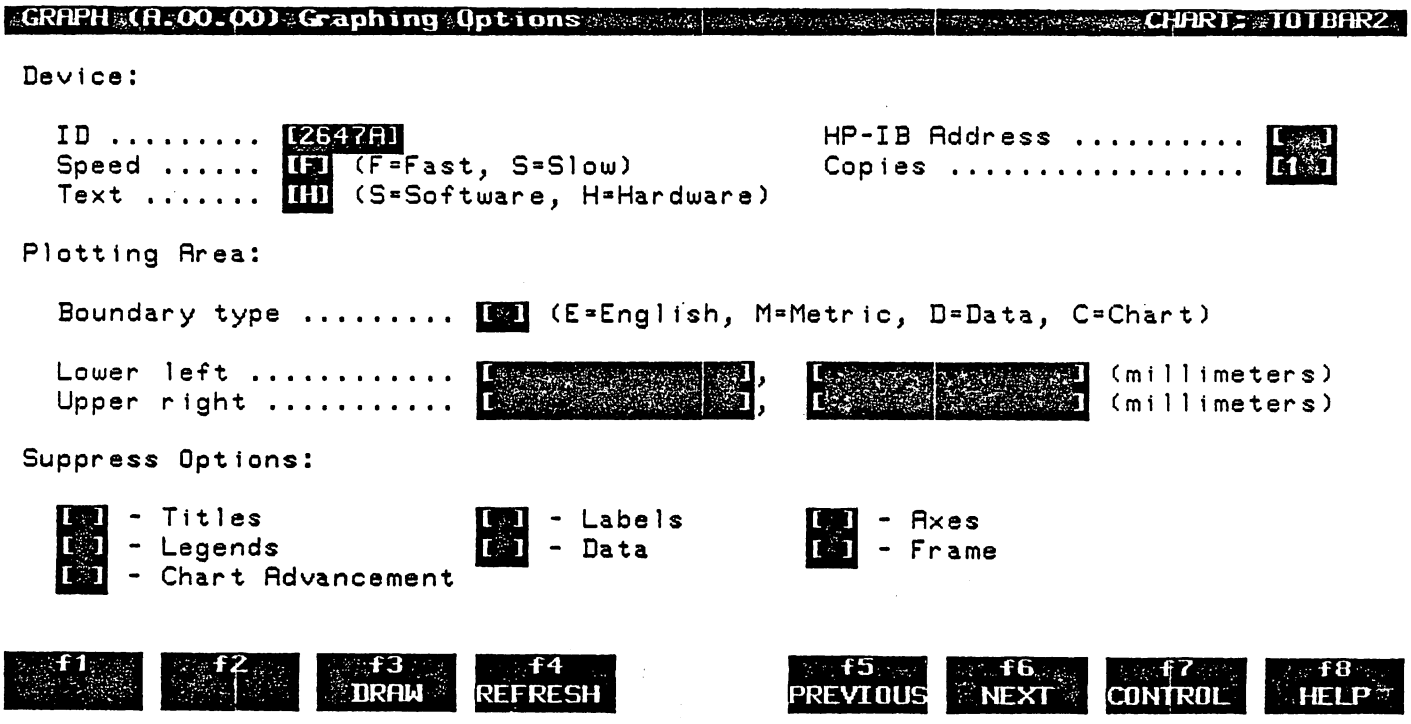

GRAPH (A.00.00) Graphing Options
CHART: TOTBAR2
Device:
ID ..... [2647A]
Speed ..... [F] (F=Fast, S=Slow)
Text ..... [H] (S=Software, H=Hardware)
HP-IB Address ..... [ ]
Copies ..... [1]
Plotting Area:
Boundary type ..... [ ] (E=English, M=Metric, D=Data, C=Chart)
Lower left ..... [ ], [ ] (millimeters)
Upper right ..... [ ], [ ] (millimeters)
Suppress Options:
[ ] - Titles
[ ] - Legends
[ ] - Chart Advancement
[ ] - Labels
[ ] - Data
[ ] - Axes
[ ] - Frame
f1
f2
f3 DRAW
f4 REFRESH
f5 PREVIOUS
f6 NEXT
f7 CONTROL
f8 HELP

Figure 3k

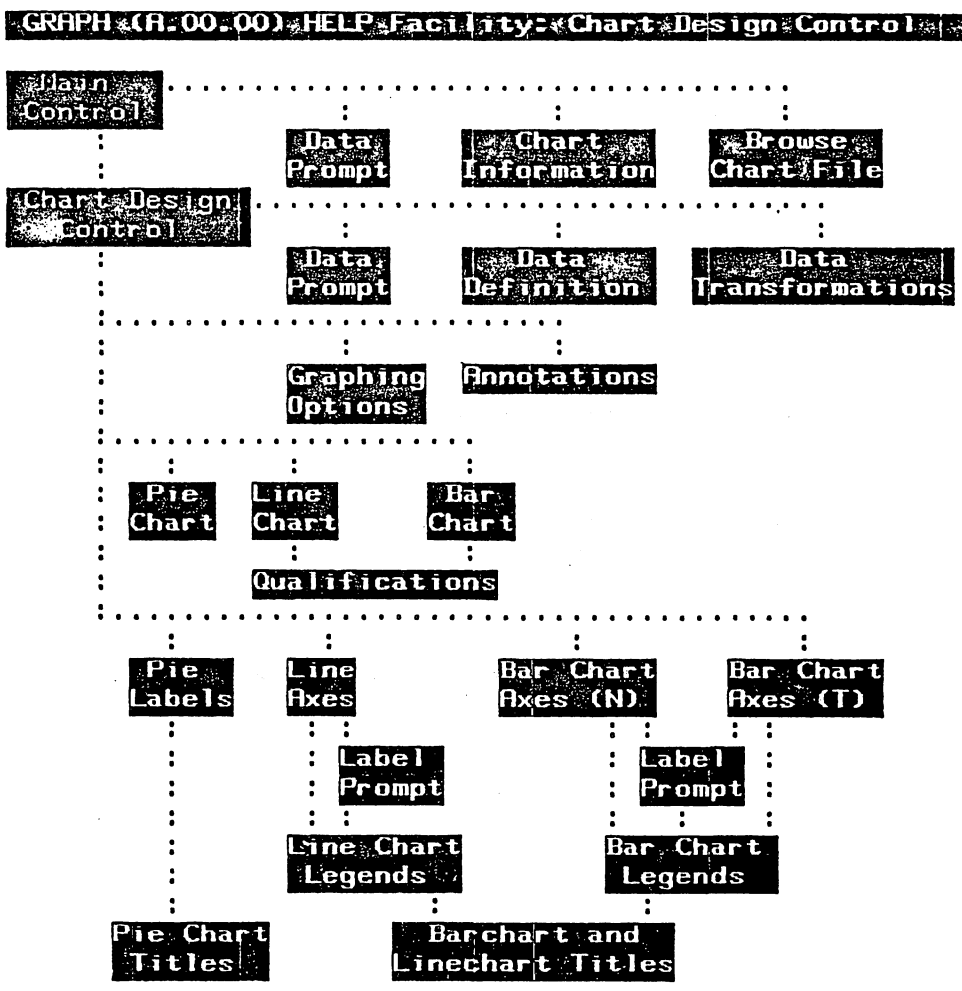


Figure 4a

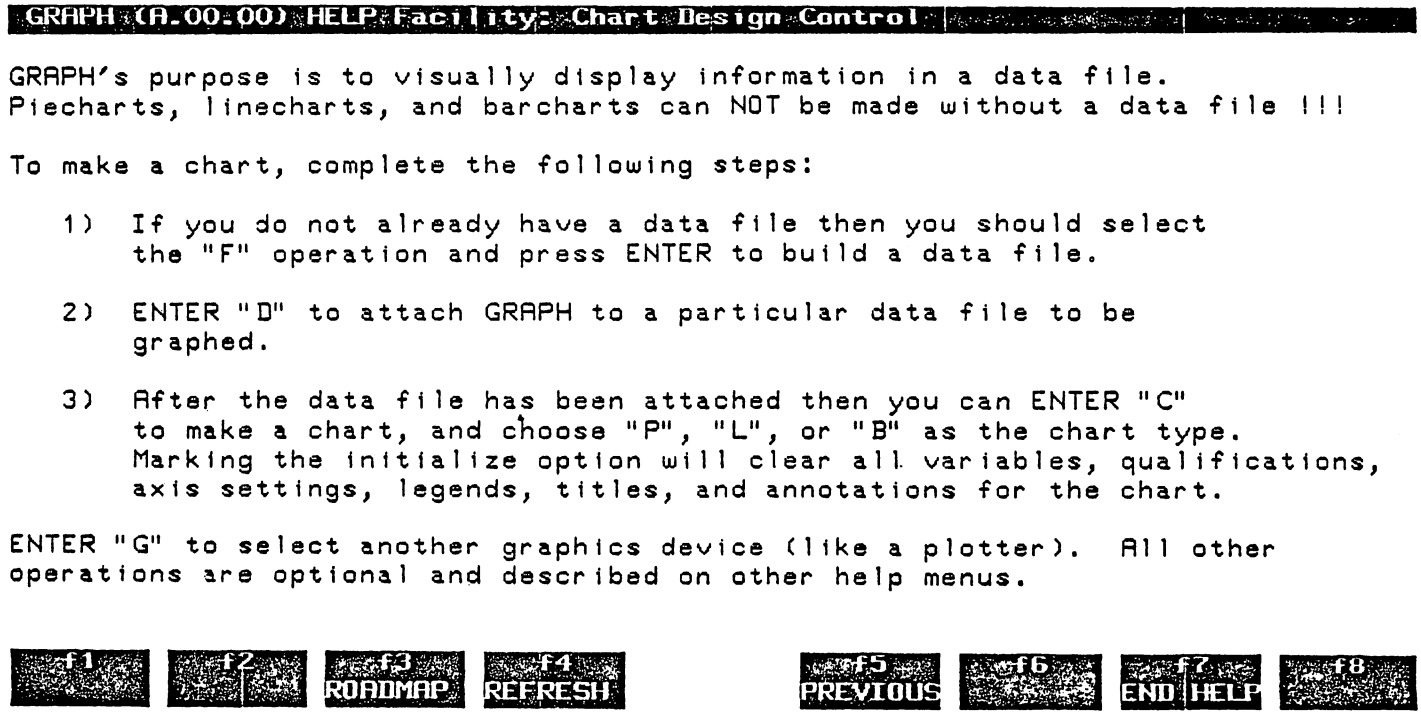


Figure 4b

Business Graphics Applications Using DSG/3000

Cecile Chi

The value of graphs as an effective method of presenting information is widely acknowledged. As a result, many graphs are being produced manually or at service bureau charges are growing rapidly. The HP3000 with the Decision Support Graphics (DSG) software package to the rescue!

Some graphics requirements are of the one-time, type-in-the-data variety, using either paper or transparencies. These are drawn by managers, professionals, or secretaries using the Multiplot package on the HP2647/2648 standalone graphics system or DSG/3000. My experience has been that people are not willing to take the time to read through the booklets and try the examples in the Self-Paces DSG course. It's always "Just show me how to start this thing!" followed by "What does it want now?" and "How do I tell it to do what I want?" Half an hour of explanation and question-answering is usually enough to get a new user going on the first application, and after that it's a matter of being available to answer questions from time to time. A major advantage of DSG over Multiplot is that neither graph design nor data disappear when the user goes on to the next graph or turns off the terminal. A one-time graph is seldom really a one-time graph. Either another copy is required, or an updated version with new data is requested, or the chart definition is used as the basis for a revised design or for the next graph to be designed. People are more productive when the graph and data are ready and waiting to be redrawn or changed.

The occasional or periodic user sometimes requests documentation of the chart design already developed and saved in order to plan the next revision while waiting for the terminal to become available, or to refer to while designing another chart on the screen. A 2631G printer attached to the 2647A graphics terminal via the HP-IB is used for this purpose. The DSG screens can be transferred to the printer by using the 2647A's Command mode. Before enabling Command mode, the cursor must be positioned in the upper left corner of the screen. Homing the cursor will get it to the first unprotected field, and using the cursor arrow keys to get it to the beginning of the window will make it possible to print the screen title. Pressing the CNTL key and the *55* key simultaneously releases the protected fields, making it possible to copy them to the printer. The sequence required to print a copy of the screen is

```
COMMAND
COPY(f2)
ALL(f3) from
DISPLAY(f3)
to HP-IB(f7)#
6 (or whatever your printer address is)
RETURN.
```

Pressing the COMMAND key disables Command mode, and pressing the CNTL key and *f4* key simultaneously turns on the V/3000 field protection, getting you back into DSG. It is necessary to get out of Command mode and into the DSG in order to get to the next screen to be printed. Paging of the printer must be done manually, since the Copy command does not cause a page eject. Two screens will fit nicely on a page, with a few blank lines to separate them. Similary, the graph drawn on the screen can be transferred to the printer using the sequence

```
COMMAND
next(f1)
next(f1)
```

```
TRANSFER(f3)
ALL(f3) from
GRAPHICS(f4)
to HP-IB(f7)#
6(printer address)
RETURN;
```

Use COMMAND to get back to DSG. The Transfer command does cause a page eject on the printer at the end of the graph. We end up with documentation which fits on 8½"×11" paper, ready to go into the user's file folder or loose-leaf notebook.

DSG contains an option (which is implemented from the Main Control menu) to draw each graph in a chart file, with or without a pause between graphs. The

graphs may be directed to either the screen of the graphics terminal or to the plotter. Drawing all of the charts from a file to the plotter requires either an operator to change the paper between plots or a scrolling option on the plotter to advance the roll of plot paper between graphs.

Another option for unattended production of the series of graphs contained in a chart file is to use the printer. Printer graphs are black and white, without anywhere near the resolution of a plotter graph, but for screening or quick reference they are frequently worth the savings in time and cost. The GRAFPRNT program on the swap tape for this conference is a generalized graph-printing program. It is run from a graphics terminal which has a graphics printer (2631G) attached via the HP-IB; it temporarily resets the device destination and chart size and then produces all charts in any given chart file on the terminal screen and then transfers them to the printer. The COBOL source code is included, since the compiled version includes directions to a 2647A graphics terminal and a printer on HP-IB address 6; these may need to be changed to fit your hardware. A pause is required to allow time for transferring the graph from the screen to the printer; the call to "FPAUSE" calls a FORTRAN subroutine which calls the PAUSE intrinsic. The PAUSE intrinsic can't be called from COBOL because it requires a REAL parameter, and HP COBOL does not support data type REAL. The 40-second pause used in this program has been adequate for all applications I've implemented so far; it may not be the optimum length of time.

The methods discussed so far have required designing each individual graph using DSG. However, there are many applications in the business world which require sets of graphs identical in design except for a few variables. One variable is usually a title line, and another would probably be either the data file name or the data subset specification. Labels and additional title lines may also be variables. An example of this method is the Sales Graph system illustrated with a flowchart (Exhibit 1) and UDC:

```
SALEGRAF
FILE TITLES=SALENAME
FILE DATAFILE=SALEDATA
RUN EXTRACT;PARM=%100000
FILE GRAFFILE=DEMOGRAF
RUN DRAW;LIB=G;PARM=%100000
**************************
ARGRAF
FILE TITLES=ARNAME
FILE DATAFILE=ARDATA
RUN EXTRACT;PARM=%20000
FILE GRAFFILE=DEMOGRAF
RUN DRAW;LIB=G;PARM=%20000
**************************
```

The title file, which may be created and maintained with EDIT/3000, contains the graph titles. It also specifies the plants for which data is to be extracted from the database and controls the order in which graphs are drawn. The EXTRACT program retrieves the specified data for the plants listed in the title file and arranges it in appropriate format for DSG. The PARM parameter on the RUN command sets a software switch which is used by the EXTRACT program to identify the account code to be used for extracting data from the database, and by the draw program to identify the chart name to retrieve from the chart file. This program may be replaced by QUERY if the database is designed in such a way that QUERY is adequate, or by AQ if it is implemented at your site and will handle your needs. The major limitation of QUERY is that it can access only one data set at a time. AQ can access and concatenate multiple data sets, and the added flexibility may make it possible to use it in place of an Extract program. Of course, if you need a linear regression on your data, you need a program. Using whichever method is most appropriate, a date file is created, containing data in DSG format.

Then a graph drawing program, DRAW, is initiated by the UDC. This program retrieves the appropriate chart from the chart file and then reads through the title file. There are optional methods of writing this program, depending upon whether the plotter being used has the scrolling option. If it is a scrollable plotter, the program can just cycle through the title file, setting the title and data subset specifications and drawing a graph for each record in the title file. If the scrolling option is not available on the plotter being used, the program is written to send a message to the terminal each time it reads a record in the title file, asking the operator whether it should draw a graph for the current title and reminding him or her to change the paper on the plotter.

Some applications consist of groups of very similar charts, such as graphs of budget versus actual over time for each of many departments or divisions, for each of a number of measures. Multiple copies are requried in many cases. Since the quantities run to dozens or hundreds per monthly batch, it is not practical to draw them on a pen plotter without a scrolling option, which requires someone to change the paper after each plot. Color plots and high resolution are not necessary for screening large numbers of graphs, looking for trends and exceptions, so the cost of a scrollable plotter would not be considered justifiable.

The next option handles the batches of graphs which need to be repeated for multilple groups and multiple measures. The method illustrated in Exhibit 2 uses data which is retrieved through RJE from a non-HP mainframe using a report tabulating utility program. The utility is run on the mainframe to produce a "printed" report according to pre-defined specifications such as no titles and the use of identification codes as column headers, with the report routed to the HP3000. Then the report is read onto a disc file using RJE. A COBOL program is used to read this disc file and flip the matrix,

producing a data file in a format suitable for DSG and printing a format listing of the data file to be used for data definition. At this point, the user designs a set of graphs in one chart file, using DSG, with the graphs in the same order as the data. The title file, containing the identification codes to be used for data subset specifications and the variable title line, is arranged in the order in which the graphs should print. Then the graph-drawing program MULTGRAF is initiated. This program cycles through the chart file, drawing a graph for each location for each chart. If multiple copies are requested, it starts over. More efficient utilization of the computer would be achieved by doing multiple transfers of each screen; in our case, it was decided to use more machine time to redraw the additional copies in order to avoid the clerical time required to sort the graphs. By printing them in the proper order, the whole run can be fed through a burster and then separated into groups for attachment to reports. Once the data file has been created and the chart designs done, any given individual graph can be selected interactively by setting the subset specification and the title on the DSG menus and then drawing the graph on the screen and transferring to the printer or drawing it on the plotter to get colors and better resolution.

Our largest monthly production run so far consists of 4 copies of graphs for 17 locations on 18 to 20 different measures, for a total of over 1200 graphs. This run takes over 24 hours, so it is either run on a weekend or run with 2 copies on two nights. Even a scrollable plotter would require approximately 5 minutes per graph, or 100 hours, which is not acceptable turn-around time. The best service bureau bid, based on large volume regularly, was $10 per graph and 4-day turnaround.

A frequent question from management or professional users is, "Can't you print the data under the graphs?"; the answer is "Well, yes, but it requires some custom programming." The method illustrated in Exhibit 3 requires two programs and two passes of the paper through the printers. The REPORT program generates reports with about eight lines of title and heading information, a lot of blank space, and then the labeled lines of data, and footnotes. This program also creates the title file and data file for DSG. If the application did not require so many heading lines, the titles could be printed from DSG rather that the Report program. The paper containing the reports is fed into the graphics printer, and another version of MULTGRAF is used to print the graphs in the blank spaces on the reports. It would be possible to draw plotter graphs on the reports instead of printing graphs, by setting the graph dimensions appropriately and changing the paper after each graph. Keeping everything in the right order would require an operator's undivided attention, however, and thus is probable not practical for production runs of any size.

The methods of utilizing DSG programatically described here are early experiments at improving office productivity by providing better tools. The fact that these tools are being used as fast as they are developed demonstrates that they fill a need, and should encourage continued development of better tools. The release by the HP lab of their Contributed Graphics Library (CGL/3000) or graphics intrinsics will provide programmers with the opportunity to develop new sets of tools.

EXHIBIT 1

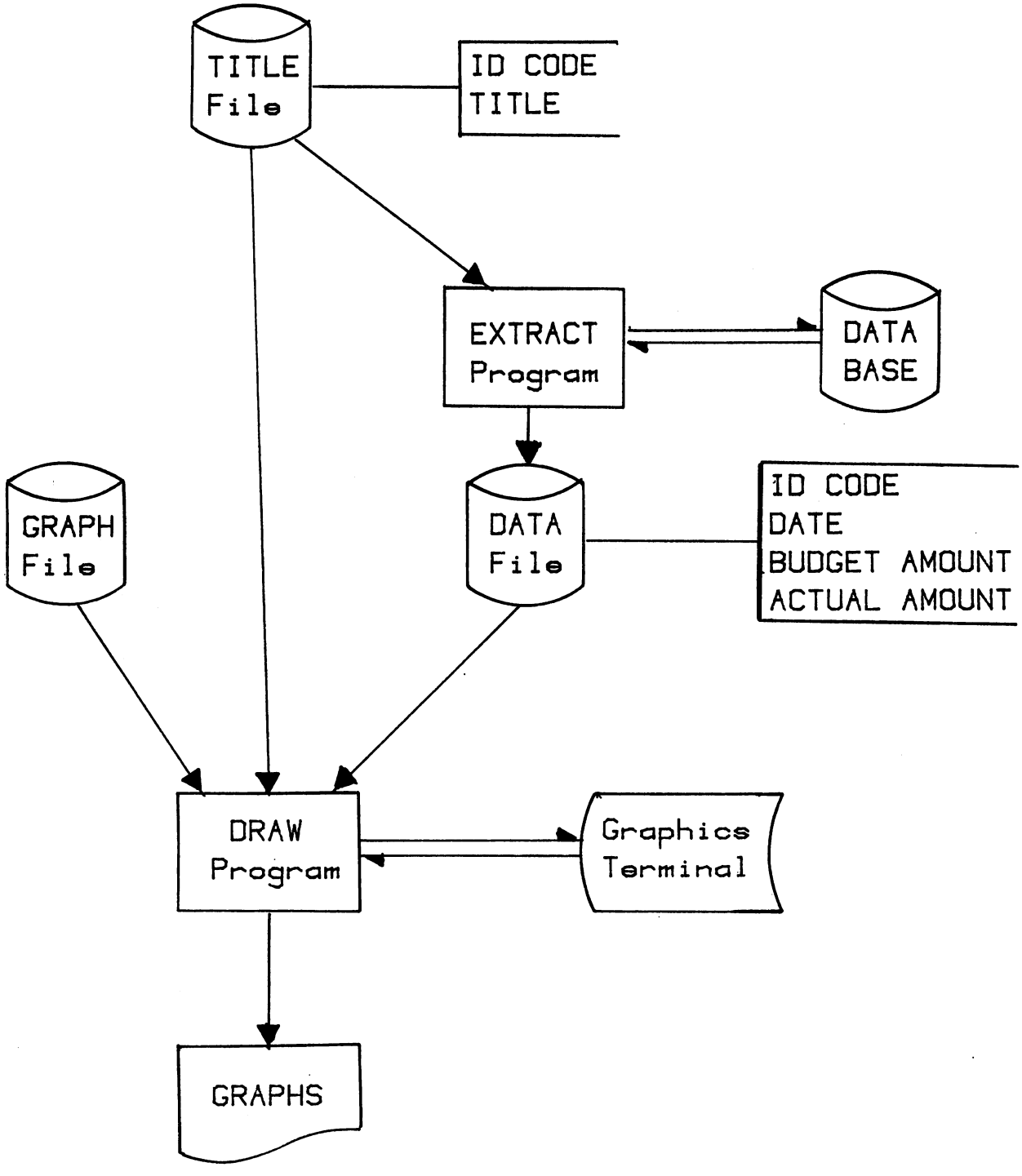

TITLE File
ID CODE
TITLE
EXTRACT Program
DATA BASE
GRAPH File
DATA File
ID CODE
DATE
BUDGET AMOUNT
ACTUAL AMOUNT
DRAW Program
Graphics Terminal
GRAPHS

EXHIBIT 2

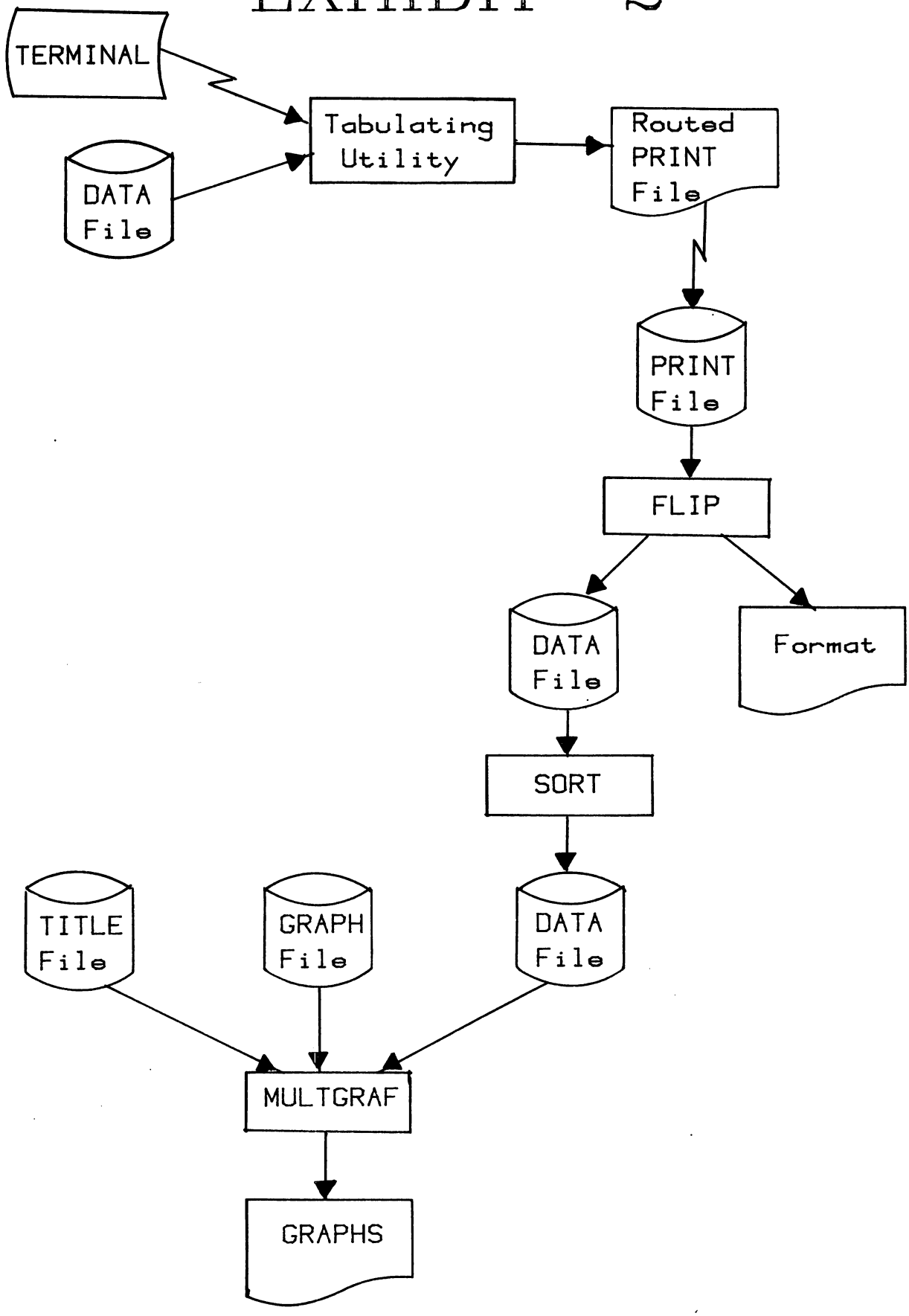
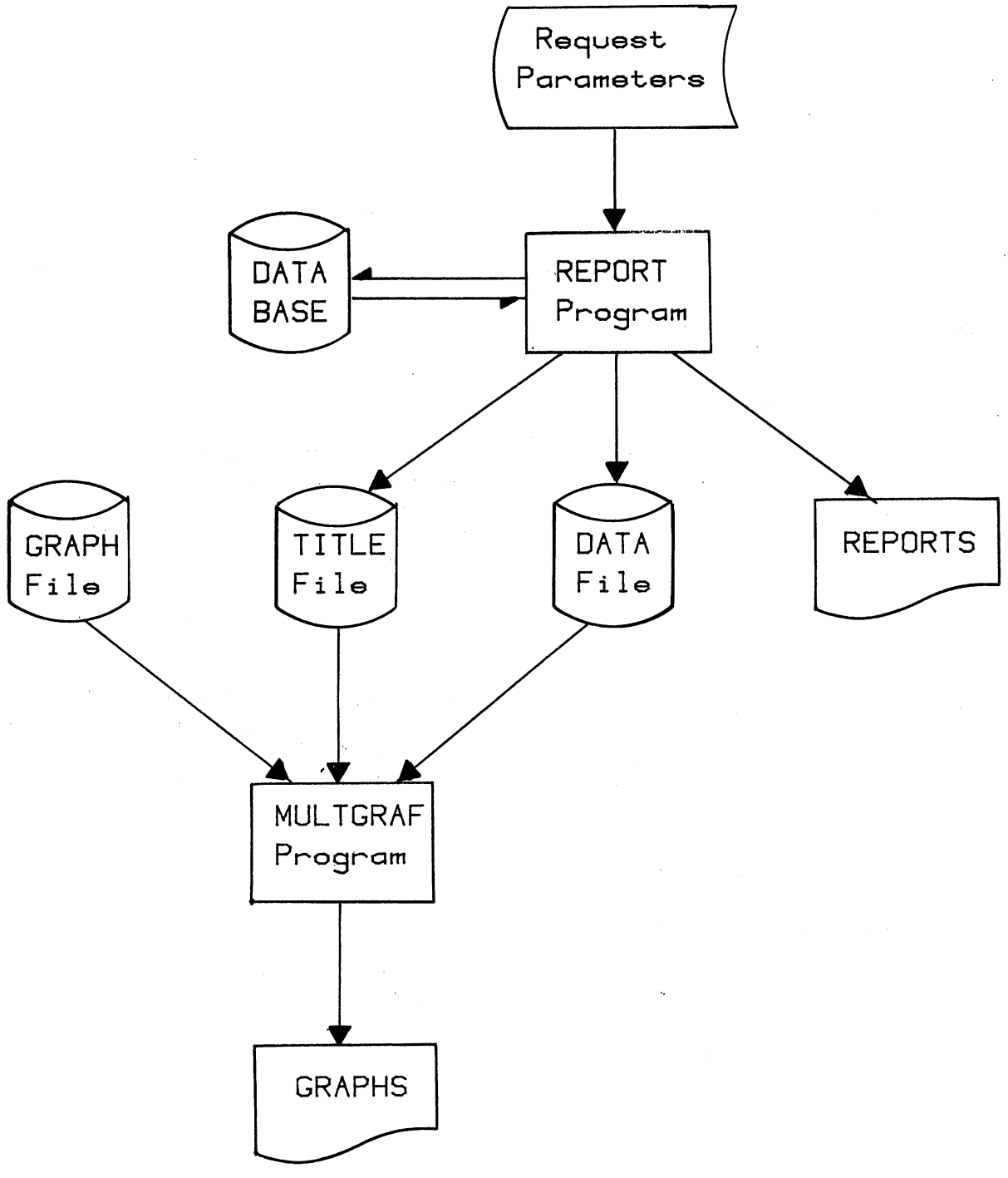

TERMINAL
Tabulating Utility
Routed PRINT File
DATA File
PRINT File
FLIP
DATA File
Format
SORT
TITLE File
GRAPH File
DATA File
MULTGRAF
GRAPHS

EXHIBIT 3


Request Parameters
DATA BASE
REPORT Program
GRAPH File
TITLE File
DATA File
REPORTS
MULTGRAF Program
GRAPHS

Tips and Techniques for Data Interface to DSG/3000

Jason M. Goertz
Systems Engineer
Hewlett-Packard
Bellevue, Washington

INTRODUCTION

In the last several years, businesses have seen an increasing awareness and use of computer generated graphics. While the military, along with the auto and aircraft industries, has en- joyed the use of computer graphics since the early 1960's, it has only been recently that graphic hardware and software has been in the cost range for the small or medium sized business. This is primarily due to the drastic lowering of the cost of minicom- puters and peripherals in the past few years. So great is the awareness and visibility of computer graphics that even Webster now lists one definition of the noun graphic as "a graphic representation displayed by a computer (as a CRT)."

Hewlett-Packard has shown to be a leader in this revolution of "affordable graphics." Calcomp and other companies have been building large bed and drum plotters (at a large cost) for many years, while HP entered the market with small bed pen plotters and other graphic output devices which cost only few thousand dollors. HP's largest plotter, while not the largest on the market, is half the cost of any plotter of comparable size and performance. Today, HP's graphic output devices take one of five forms:

1. Digital 8 pen plotters, such as the 7221C and 9872C.
2. Digital thermal printer-plotters, such as the 7245.
3. Raster hardcopy devices, such as the 7310.
4. Raster CRT devices, such as the 2647, 2648, 2623 terminals.
5. Desktop computers with raster CRT output, including the 9845C, with color CRT graphics. We will not deal with this in this paper.

Digital devices are those that receive data in a digital form, such as characters sent via a modem, or HP-IB link. This is different that an analog device, which plots databased upon changing voltage, current, or some other data source whose range of values is a continuum, rather than discrete points. Raster devices are those whose graphic output is formed by a matrix of dots, either turned on or off. All HP CRT terminals use raster technology. If the reader is unfamiliar with these terms, a very good computer graphics "primer" is available from Hewlett-Packard, called Becoming Comfortable with Computer Graphics. This is a small document published by HP's San Diego Division.

Computer hardware is, of course, useless without software to drive it. Hewlett-Packard's graphic software offerings are many and varied, with software available on all of the computers built. HP has even written it's own graphics languages, such as HP-GL, which most of the 8 pen plotters use as an internal language, and AGL, a high level language which is standard among many of HP's computers. HP has used these languages to write several applications, such as the graphics package on the HP1000 ("BRUNO"), and a package, aimed at the business market, which runs on the HP3000, called Decision Support Graphics, or DSG/3000.

DSG/3000 is a high level application software package. It is optimized to easily define and generate graphic plots that are commonly used in the business world. These plots are line charts, scattergrams, bar charts and pie charts. DSG is also capable of producing slides plots with only textual data, commonly referred to as "slides." DSG is not capable of performing so-called "technical" graphics functions, such as computer mapping, Computer Aided Design, or real-time graphics. However, the functions that DSG is designed to perform it performs very well, and is easy to learn to use.

The primary advantage of DSG is that it runs on the HP3000, and thus has access to all of the data storage capabilities of the HP3000. This is ideal for business graphics, which usually deals with sales figures, forecasts, budgets, and other data that a business would normally store in a computer. The purpose of this paper is to show how DSG can be interfaced to the data storage technologies available on the HP3000, and thus utilize the full potential DSG to help a business's managers make decisions. While this paper is not intended to be a primer on the use of DSG, a few of the functional aspects of the package will necessarily have to be mentioned. For those who would like to learn more about the actual use of DSG, the reference manual is excellent, and a self study course is available.

DSG'S VERSION OF THE GRAPHICS WORLD

DSG defines everything it does in terms of a chart. Charts are contained in a file called a Chartfile. It has a unique file type, and is known to MPE as the file type GRAPH, and file code 1083. This is directly analogous to the way VPLUS/3000 deals with forms within a forms file. The program GRAPH.PUB.SYS is used to define all of the characteristics of the charts within the file. These characteristics are all entered via menus (screens), and include things such as what type of chart it is (bar, line, or pie), what color pens will be used to draw the chart, how the axes will be labeled and scaled, etc. One other characteristic that is associated with the chart is the name of the data file that will be used in the plot. Currently, the only means of data input into the plot is via this file.

In addition to defining the name of the file, we must also describe the contents of the file. Both of these functions are done with the Data Definition Menu in GRAPH. In this menu, we define what the variable names are (to be used in another menu), what columns (or field number) of the file the data is in, and whether the data is numeric or textual. Up to eight variables can be defined on this menu. Every type of chart (bar, line, etc) has a specific menu associated with it. It is on this menu that the specific variables that are to be plotted are defined. Only those that are to be plotted need be defined, even though the file may contain more. This implies that one data file can be used for multiple charts. All that has to be done is define different data items to be plotted, but the same data file name.

The question at this point is how do we put data into this file? Really, this is the issue that this paper deals with. There are as many ways to create this file as there are programs and programmers. We will deal with a few of the basic ones.

First, the GRAPH itself allows the user to build a file of data. In fact, this is in some ways the most convenient way to build the file, since GRAPH builds what HP calls a self-describing file, a file type that will be used more and more in the future. This is a type of file in which the file itself contains the informaton just mentioned, that being what the variable names are, what type they are, and where in the record they are located. These data are contained in the user labels of the file, and are defined by using the Data Prompt Menu of GRAPH. This allows the menu screen to automatically define the data items to be plotted as soon as the data file name is keyed in. While this may seem the best way to define the file, there are a few drawbacks. First, all the data must be entered into the GRAPH screen. Therefore, if the data resides in another file structure, the data must be listed and keyed by hand. Only one screen of data can be entered, thus limiting the number of data points to twelve. In addition, only five variables can be en- tered on this screen. If the chart is going to be used repeatedly, a lot of time will have to be spent keying the data into GRAPH. This menu is very useful for graphs in which there is a small amount of data, the data is not resident anywhere else in the computer, and the graph is going to be used only once or twice.

For charts that are going to be used only once or twice, but requires more than twelve data points or five variables, the next best method for building the data file is to use EDITOR. HP's editor can be used, or any of a multitude of other text editors and word processors available, such as QEDIT, QAD, EDIT2, TDP (LARC), HPWORD, HPSLATE, etc. While this will not build a self describing file like GRAPH, it is a cheap way to build an ASCII file of data. Every system has at least the HP EDITOR, and most have at least one other. It is only a matter of learning how to use whatever editor is desired, and build a file. With this method, the Main menu for each type of graph will have to be used to define the data items, lengths, etc.

Another software package that can be used for this function that virtually every site has is VPLUS/3000. It is a relatively simple matter to define a form with fields for the desired data. Using ENTRY, this form can be used to key the data into a Batch file, then reformatted to a DSG compatable format by using REFSPEC and REFORMAT. All of these tools are currently provided with every system as part of the Fundamental Operating Software.

Many, if not most, of the charts that are defined are used more than once. Usually, the charts are used on a periodic basis to plot data. For example, a bar chart of monthly sales for the last 12 months is generated every month with the current month's data added and the data from a year ago deleted, or a pie chart showing expenses for the last quarter. When this type of chart is used, it is often desirable to extract data directly from the files which contain them, such as an IMAGE database or KSAM file. The trick is to get the data from the native data struc- ture into Graph's sequential file in the format and record locations defined in the chart.

If the data file is stored in a sequential file or a KSAM file, and if it is all stored as ASCII characters (as opposed to binary data, such as COMP or COMP-3), then DSG can directly access the data file. All that is necessary is to define the sequential or KSAM file in the Data Definition Menu, and process the graph. However the file will, most cases, contain too much data for one plot. For instance, a sales file might contain not only a record of total sales for a product for the last month, but also a record of every sale. It is not desirable to plot every sale record, but just the summary records. This can be done by using the Data Subset Specification on the appropriate chart Main Menu, which is the same menu used to specify which variables are being plotted. Perhaps a better way is to FCOPY the records into another, smaller, sequential file. By using the ;SUBSET option of the FROM=;TO= command, the proper records can be chosen. The file that these are copies is the one defined on the Data Definition Menu.

The most common form of data storage on most HP3000's used today is the IMAGE database. Indeed, many people buy the 3000 just for this feature, and use it heavily. Since the database files are privileged, and are formatted in a very special way, one cannot extract data from them quite as simply as from a sequential or KSAM file. Either a program must be written using IMAGE intrinsics that will extract and format the data, or an existing report-writing program must be used. HP currently has three of these report-writers: QUERY, INFORM and REPORT. INFORM and REPORT are part of the 4 module system called RAPID/3000. At this writing, REPORT and INFORM are announced but not yet released by HP. Therefore, they will only be mentioned here. There are currently many others that are marketed by OEM's, third parties, and software houses. Among these are ASK, QUIZ, AQ, and REX, to name a few.

It should be mentioned that there is a contributed utility call DB2DISK that takes data from a data set and writes it, as is, to a sequential file. At this point, the data can be treated like any other sequential file. However, DB2DISK does no formatting of the data, and if any of the data is in binary format, DSG will not be able to utilize it for a plot. Also, because of the nature of an IMAGE database, desired data will reside in multi- ple data sets, and DB2DISK will not be able to combine the data in the desired fashion. Therefore, DB2DISK can be useful if all the desired plot data resides in a single data set.

The main problem with using these programs is that most are designed to write a hardcopy report. Few, if any, are designed to write to a sequential file. The trick is to get the report writer to think it is generating a printed report, when it is really writing to a normal MPE file. Fortunately, this is made very easy by the nature of the MPE file system.

Any one of the report writers mentioned above always writes its report to an entity that MPE defines as a "file." A file is a "hole" that a program either reads data from or writes data to. The program, when opening the file, asks for the file to reside on a specific hardware device, such as a terminal, line printer, or disc. After the open, it reads and writes data normally. As long as the data looks correct, the program is satisfied. It is possible to externally redefine the device on which the file resides, along with many other file characteristics such as record size and type. This is done via the MPE :FILE command. The main bit of knowledge that must be gained is the name of the file that the report writer uses for its output. Once this is known, it is a simple matter to define a :FILE equation which will redirect the output file to a disc file. This can then be used by DSG as has been discussed before.

The following is an example of how the output of QUERY would be redirected to a DSG sequential file:

```
:BUILD DSGFILE;REC=-100,5,F,ASCII;DISC=1000

:FILE QSLIST=DSGFILE,OLD;DEV=DISC

:RUN QUERY.PUB.SYS

>OUTPUT=LP
>XEQ DSGRPT      << GENERATES REPORT TO GRAPH
                    DATA FILE. >>

>EXIT
```

The main point to be noted is that the file name that QUERY uses is QSLIST. This is necessary for the :FILE equation. The file must be defined as being on Device=Disc, as QUERY will default to a line printer, and that is an OLD file. Every other report writer has file name, that would be used in the same fashion. Also, because QUERY will attempt a page break after 60 lines, the NOPAGE option must be used in the report. The reference manual for that report writer should be consulted for this type of information.

All of the above methods are, of course, dependent upon the format of the application's data structure. If the file has binary data, or if no summary records exists, then a program will have to be written to extract and summarize the data to a form that DSG can read. Since this requires a great deal more effort than using QUERY, FCOPY or some other existing utility, things can get a bit more complicated. Many shops have a degree of red tape that must be dealt with in order to get a program developed. Whether or not this type of effort will be required is a very important consideration when developing a graphics application.

TREATMENT OF DATA

Before concluding, one thing should be mentioned about data treatment. It has been said that one can prove anything with statistics, and this adage is true also in business graphics. It is very important that a business manager understand exactly what he is looking at. In other words, what a chart says can often be misinterpreted because the person looking at it does not understand what the numbers on it mean. An example of this

could be a chart of net change in sales dollars. Assume that sales had been growing by 100K every month, from 1.1 Million to 1.7 Million in 6 months. The net change is what was plotted. If this data was read as sales dollars, the person seeing this plot would think that the company had only sold 100K each month, when it had been selling close to a million. This is, or course, a somewhat absurd example. But the point is that represented data must be clearly labeled and clearly understood by the intended audience, or the plot is worse than meaningless. It can be disasterous!!

CONCLUSION

We have seen that DSG can utilize the data stored on the HP3000 very well. All that is necessary is to put the data into a format that DSG can read, that format currently being a sequential file. All that is then necessary is to define the data file name and the data variable names, along with the type of data and locations within the record. With this method of data interface, DSG becomes a very powerful tool, giving managers graphic representation of what their business is doing, thus allowing decisions to be easily made.

Project Management with the HP3000

Nichols & Company

N5500 is easily the simplest, most convenient and most flexible Project Planning and Control System available for the HP3000 user community. N5500 offers three levels of planning and control: strategic for the construction industry and other large scale efforts, tactical for manufacturing and plant maintenance and dispatch for individual professional efforts in such fields as Engineering, Research and Development, and Management Information Science.

N5500 is the third fully implemented project planning and control system developed by Nichols and Co., Inc. The product has been in use for more than five years by over two hundred and fifty companies. Over the past ten years, Nichols & Company has alone invested more than fifty man years in N5500 and its related options. The Company's corporate office is located in Culver City, California just minutes north of the Los Angeles International Airport. It has branch offices in New Jersey, London, Manila, Mexico City and Stockholm.

N5500 was introduced after an intensive analysis of current industry needs. It is a highly versatile, yet extremely easy to use package. There are no redundant input formats or complex requirements. It is simple, direct and logical. But N5500's greatest strength is its versatility of application and overall usefulness. Nichols and Company has built and fully supported N5500 to make it a viable tool for all project oriented groups.

MANUALS. N5500 instruction materials are complete but not overwhelming. Our manual is clear, concise, easily understood and logically presented. It starts from the general and moves easily to the detailed, using frequent illustrations and meaningful examples. It contains a table of contents and a complete index. Terms and concepts of project management are defined and illustrated. Though five full days of training and assistance are provided with N5500, several clients have been able to implement the system and use it successfully with no vendor assistance, other than the information provided in the manual and installation guide.

EASY ENTRY OF DATA. Whether in interactive or batch mode, N5500 offers easy and simple entry. There are no redundant entries, complex codes or difficult manipulation. For planning, there are only two input formats, where other products range from eight to over thirty. Any number of standard projects can be preloaded and recalled at any time. This repetitive approach can save days of effort and prevent coding errors. All standard resource data can be preloaded and automatically allocated to incoming plans as needed. If the interactive entry option is used, data is validated and fully edited on input. Errors are identified and corrections are made easily.

EASY RETRIEVAL OF INFORMATION. Again, whether interactively or in batch, N5500 is superior to others. Accurate information is easily accessible via standard or custom formats. But even the standard reports have countless variations that support a wide variety of information needs. Individual reporting variations can be preloaded and called out at will. Support schedules can be established to suit an organization's recurring normal requirements based on any given variance such as weekly, monthly, quarterly or yearly cycles.

FULL PERT/CPM CAPABILITY. Precedent networking is standard. Node (I/J) processing is optional. N5500 produces late start, early start, expected start, late finish, early finish, expected finish, and negative float calculations. The system also enables the planner to specify varied task relationships such as start to start, start to finish, finish to start, finish to finish, start plus/minus days, finish plus/minus days and spacers. Tasks can be scheduled at any level of effort in numerous periodic variations such as once weekly or once monthly. Unlimited weekly cycle and holiday calendars can be applied. Critical path analysis is standard and is constantly recalculated to portray a changing environment.

GRAPHICS. It is usually easier to interpret an illustration. Graphic representation of digital data allows the interpreter to see ranges and variations in better perspective. Nichols' product produces graphic information on both line printers and graphics plotters. Printer information includes bar charts and trend analyses. Plotter displays report the intricacies of a network, especially over extended periods of time and highlight the critical path directly.

SIMULATION. Whenever a plan is entered into N5500, the system instantly identifies and highlights the conflicts and discontinuities of schedules, as well as providing an accurate cost analysis. The plan can be simulated as a stand alone element, or just as easily as a dynamic part of the inprocess load. In other words, N5500 can show the impact of adding one or more projects to the current schedule and workload without permanent modification of an inprocess load. It will display displacement and overload situations easily.

CAPACITY PLANNING. The awareness of what resources are needed to support and accomplish either a simple or complex workload is essential in today's ever changing environment. N5500 tells when work will be

accomplished if there is a limitation in available resources and suggests what resources are required to meet schedule demands. Mixed variation on both themes are easily established using selected force loading options or by force loading the critical path. When load leveling, N5500 uses the significant accuracy of a daily loading pattern without any additional demands on the planner.

PROGRESS REPORTING. Most systems fail completely when an easy and routine mechanism for periodic maintenance of project progress is not available. N5500, however, automatically provides a special turnaround document or video screen for the entry of project accomplishment. The hours or days worked may be entered as individual or group efforts in any increment from tenths of hours to man weeks. Money, materials or other units consumed may be reported also. This progress data automatically updates the network and produces a series of forecasts and an "earned value" calculation. N5500 even produces a detailed analysis of estimated versus actual performance on a project, dividing the work into classes or types and reporting it across projects by assignee.

ACCOUNTING. Once actuals are assimilated and validated, a wide assortment of reports can be generated. N5500 handles labor, materials, equipment and cost accounting easily and quickly. It provides both gross and net continuous expenditure reporting by responsibility area or by type of effort. Invoicing can be handled directly from the system. Cost data from other systems can be integrated easily into N5500. N5500 can feed general ledger or journal entries. Actual versus planned expenditures over days, weeks, months and years can be readily assessed.

EDUCATION. Nichols and Company believes well trained clients are the most successful users and successful users profit greatly from planning with N5500. Upon purchase of N5500, you receive five days on-site training at our expense. We want your N5500 system running smoothly. Many organizations find that ongoing education enhances the effectiveness of managers and planners alike. We therefore provide General Project Planning and Control Concept seminars. This continuous belief in education helps people live up to their true potential. They should never lack a basic understanding of the project management discipline. We also provide tutorial and general consulting services for those organizations who simply lack the appropriate manpower to monitor their projects. At each point, we are at your side, to provide whatever is needed to make your project management effort a success.

SERVICE. Closely allied to education, is Nichols and Company's overall service orientation. Our consulting and technical staff are readily available by phone in either our New Jersey or California offices. If needed, we provide immediate assistance to help in an emergency. We provide computer backup, plotter services and technical advice. For those organizations, not yet ready to install their own N5500 system, we have several outstanding time-sharing services with dial-up access to N5500. Finally, for those who want to assess N5500 fully and gain hands-on confidence, we conduct free monthly workshops. At these workshops, you bring your projects and load them into N5500 yourself to see how N5500 can work for you throughout the project management process.

FINALLY. N5500 is a full capability product. It can be used in any environment from Construction to Engineering, Manufacturing or MIS. The system provides Strategic (linear projecting), Tactical (group loading), and Dispatch (individual loading) planning in an easy to learn, easy to use format. The product is known for its simplicity, convenience and flexibility. It has more useful features than any other project management system in the HP3000 market today.

Using the HP3000 for Decision Support Systems

Robert Shelley
Noesis Computing Company

WHAT ARE DECISION SUPPORT SYSTEMS?

Decision Support Systems have been around us all for many years, but it only recently that we have been using the term. I believe that DSS appropriately describes the management oriented use of interactive databases, financial models, graphics and statistical analysis in meeting the day-to-day information needs of decision makers. The principal characteristics of a DSS are:

- It is in an interactive or on-line environment;
- It can be developed (at least the first phase) in less than a month.
- With minimal training, it can be maintained by the end-user.
- It does not post to a general ledger.
- It allows the user easily and quickly to change assumptions about the data and produce reports reflecting those changes.

I think that "Management Information System" was originally intended to describe what is now referred to as DSS. But the problem with MIS is that the term grew in usage and scope over the years to the point that it no longer really means anything. To wit, a manager of MIS is often responsible for all computer operations, programming, technical support, and most recently even for word processing! Thus the need for a term like DSS which is more restrictive and focused.

Typical DSS applications are:

- Customer, employee, or equipment tracking systems
- Cashflow projection systems
- Merger/acquisition models
- Budgeting models
- Detailed analyses of particular functional areas (e.g., A/P, A/R, Foreign Exchange)
- Dun and Bradstreet marketing databases
- And much more.

HOW DSS DIFFERS FROM TRADITIONAL DP AND WHY

The orientation of data processing has traditionally been toward operational control and audit trails, and this is as it should be. These controls imply the need for systems development projects with a great deal of planning and structure, as well as an emphasis on efficiency of processing due to the volume of data that has to be handled. The tools and techniques used to meet the needs of transaction processing systems are dictated by the objectives of these systems.

But by their very nature tools like COBOL and techniques like batch processing are not adequate to meet the information needs of a mid-level or senior manager. Typically a manager needs information on an ad-hoc basis in order to help solve a problem (i.e., put out a fire) or to investigate an opportunity. For this kind of objective, higher level languages than COBOL are needed and the computing environment has to be interactive. Ideally you should be able to access an existing data file and select out and report on specified subsets of data. The emphasis here is on getting out only the relevant data; how it is presented is less important. You should also be able to create an ad-hoc system to address a specific question, knowing that the system will be thrown away after it has done its job. The whole cycle could be less than a week.

In summary, because the objectives and needs of managers are so different from those of operations staff, it follows that the types of programming languages and the computing environment chosen to meet their goals will also be different.

THE IMPORTANCE OF DSS USER SUPPORT

If and when an organization begins using Decision Support Systems, it will be necessary to develop a support program that can meet the on-going needs of the DSS user community. This program includes: training users, addressing day-to-day questions of users, providing access to the computer for new users, and more. Any part of this (or even all of it) can be achieved by using resources outside of your own organization.

User training classes should be no more than two days long. This is because few users have the time to break away from their regular activities for more than two days at a time, and also because, in my experience, if the software product cannot be taught in two days then it is not a user-oriented DSS product. At the end of the class users should be able to return to their offices and start producing simple reports, models or graphs. User training is often provided by the company selling

the software (e.g., HP or a software house). Sometimes classes are available locally which makes user training easier to fulfill. When local training is not available then the choices are sending a user to a training class at another location, bringing a trainer into the organization to train a group of users, or developing a training capability within your organization. A quick cost/benefit analysis will lead you in the right direction.

Since time is often a criticl factor when a DSS system is used, a responsive user assistance program is most important. This can be provided either by the software vendor or from within an organization. The choice is most often dictated by the number of DSS users within an organization. Because DSS users are generally outside the DP area, the people providing DSS user assistance need somewhat different capabilities and backgrounds from what you might expect in a data processing support group. They include:

- An ability to avoid computer jargon and communicate effectively with those who know very little about computers.
- Some background in the business side of the organization. This can be either from work experience (in the case of someone who is interested in transferring into a systems department), or through education (for example, a programmer who has gotten an MBA degree).

  User support also means insuring that new DSS users have access to the computing resources. This includes assistance in getting the right terminal for the intended application and when necessary working with the phone company or any data common-carrier in setting up a leased line or dial-up access.

WHY THE HP3000 IS GOOD FOR THE DSS-TYPE SYSTEMS

Stated simply, the HP3000 is an excellent choice for a Decision Support System. The computing environment and operating system of the HP3000 meet one of the fundamental requirements of any DSS — they are configured to make interactive computing straightforward and easy for a user.

Just as important, both HP and others offer good DSS-oriented software for the HP3000. In reviewing the available products, I will categorize them into four main groups: database management, financial modeling, graphics, and statistical analysis. As you know the IMAGE database management system provided by HP with all HP3000 computers is an excellent DBMS product. When IMAGE is coupled with a user-oriented report writer and screen generator like RAPID/3000 or the products of Quasar Systems (i.e., QUIZ and QUICK), a user can easily do the kind of ad-hoc reporting that is characteristic of any DSS.

Financial modeling software has been around for quite a while, though it has only recently gained a great deal of attention due to the success of VISICALC, a popular personal computer software product. HP does not offer a financial modeling product at present but there are a growing number of software houses that do. Among the products I am aware of are Dollar Flow, the Interactive Financial Planning System (IFPS), and EPS-FCS. Most of the providers of this kind of software will have an information booth at this conference.

Almost all of the currently available graphics software comes from HP. Decision Support Graphics (DSG) is a very user-oriented package that makes it quite easy to get out meaningful graphs with minimal effort. PLOT/21 is another HP product but is really a set of program callable subroutines that allow complete control over a plotting device. PLOT/21 is essentially a programmer's tool. And then there is SMOCK, a graphics product which is generally adequate for many users and is available from the HP Users Group Contributed Library.

In the area of statistical analysis software, there are three widely used products for the HP3000. Statistical Package for the Social Sciences (SPSS) has been on the market for a long time but is batch oriented. A product called IDA was developed by the University of Chicago Graduate School of Business and stresses interactive usage, forecasting, and regression. Also available is the BMDP statistical package from UCLA which is batch oriented but good at multivariate techniques and analysis of variance.

FUTURE TRENDS IN DSS

Looking at the future, there are a number of developments that will have an impact on the DP and DSS user communities. Among them are:

- The increasing presence of personal computers;
- The sharing of data between various parts of an organization;
- The growth of electronic mail;
- The migration of DSS languages and techniques into the DP community

With this audience I am sure I do not have to discuss at length the recent growth of microcomputers — we have all seen it first-hand. However there are some subtler changes taking place that deserve to be mentioned. Microcomputers are rapidly accomplishing what so many of us have tried so hard to do for many years, and that is to demystify computers and programming. A single product — namely VISICALC — has unleashed the imaginations of uncounted managers and analysts. The challenge for us is to maintain their enthusiasm and at the same time ensure that their expectations are in line with what is possible.

Sharing data, as well as the responsibility for maintaining it, can provide significant savings to any organization. As you know, database management systems have done so much to eliminate redundant data among different files by using keys to facilitate sharing data. In the same way we are also seeing a growing need

for departments to share databases instead of each department expending the time and effort to maintain its own version of identical data. And the security features of DBMS and screen handlers make this sharing possible even when one department's data must be kept entirely within the department's control.

Electronic mail is usually associated more with "offices of the future" than with DSS, but in my experience the same computing environment that supports DSS should also be able to support electronic mail requirements. Until recently I worked for a large commercial bank that had an electronic mail product available on its interactive computer. This computer was accessible from all major offices throughout the world. On a daily basis I was in touch with five distant groups that I was managing. I can assure you that having electronic mail available made it possible for me to keep on top of what was happening in the field and address any issues as they surfaced. Also, many of the executives of the bank who traveled extensively found that electronic mail was the best and cheapest way for them to stay in contact with their offices and with others who needed to reach them.

Lastly, I think that in the long run there will be a number of tools and techniques that will migrate from the DSS environment to the DP environment. As they mature and grow, user-oriented report writers are already being used by both DP and DSS users who find them so successful in increasing productivity. Another example is the building of a prototype system in the earliest stage of systems development. This concept is far from new, but it is my observation that it is more consistently used in developing DSS applications than in DP. Because the use of a prototype is fundamental in insuring that user and developer understand each other, I hope that this technique also will find its way into more and more DP projects.

CONCLUSION

The HP3000 provides an excellent computing environment for Decision Support Systems, when coupled with the proper user-oriented software. The discussion above has tried to bring together what DSS is, how it is being used, what software products are available, and some of the issues that can affect DSS users and providers. Other software products and other issues are surely out there. In order to provide a forum for them, a new Special Interest Group — SIGDSS — is being formed now and will meet for the first time at this conference. Look for the meeting time and place in your conference schedule. If you cannot attend but have an interest in the group please contact me for further information:

Bob Shelley
Noesis Computing Company
615 Third Street
San Francisco, CA 94107
(415) 495-7440

The Truth About Disc Files

Eugene Volokh
VESOFT Consultants
Los Angeles, California

I/O, I/O, it's off to disc we go . . .
(modern rendition of Walt Disney)

ABSTRACT

The disc file is probably the most important part of MPE; however, due to the large number of different options and considerations inherent in disc files, these objects are often "under-understood" — this paper will try to present the truth and nothing but the truth (the whole truth will not be printed owing to lack of paper) about disc files, which will hopefully remedy this situation.

CHAPTER I
FILE STRUCTURE

Where It's At

Before discussing disc files themselves, we must take a moment to point out some terms, probably already known to you, regarding the physical medium on which disc files reside — the disc. This disc consists of a lot of 128-word SECTORS, and is assumed to be configured on the system as one logical device.

Some considerations to be judged when referring to these discs are: (1) space — each disc has an ever so finite amount of sectors on it, the number of which varies from disc to disc, but is, by Murphy's Law, never enough — and (2) speed of access, which is typically on the order of 30 disc accesses per second.

The discs typically used with the HP are ones that constantly rotate in order for all parts of the disc to be accessible by the unrotating DISC HEAD. It is this rotation of the disc that is the culprit in the slowness of disc accesses. Similar considerations can be applied to the two other significant types of hardware: memory (which is very, very fast yet lamentably limited — up to 4 MegaBYTES on a Series 44) and tapes (which are virtually infinite yet quite slow).

The above hardware considerations, though elementary, will be of paramount importance in further discussion.

The Extent Question

Let us start at the beginning — the creation of the file. We will examine what the MPE operating system has to do to create a file. For example, let us say that you ask MPE to build you a data file, which is to have room for at most 100,000 records of 128 words each (note that 128 words is the size of a sector, and thus a good value for simplicity). This would be done, perhaps, by an MPE command akin to ":BUILD ING;DISC=100000" (MPE will automatically assume 128 words as the record size). Now, what does MPE do?

Well, of course, MPE must allocate some disc space for that file. In this particular case, MPE must allocate a whopping 100,001 sectors (the 1 extra sector is for the file label, a place where MPE holds internal file information like the lockword, etc.) all at one time. But, wait a minute! There may be 100,001 sectors out there on your disc (or discs), but it's possible that there is no one single gap that large out there. Moreover, maybe you don't really need all that space. Quite probably, you'll never use more than 10% of it! So, we are faced with a dilemma — if MPE were to allocate the space for that file nicely and simply, in one big chunk, it may not have enough space on disc; or, if it does, most of that space will probably be wasted, as (for a time, at least) you will not use all of that space.

Let us look at the other "extreme" solution. Why don't we, perhaps, allocate only one sector of space at a time — one in the beginning, for the file label, and one every time the user needs one. That way, even if the disc is hopelessly fragmented (i.e., there are very many 1-sector pieces of free space out there, but no large ones), we can probably fit a sector — if we can't, time to buy another disc; moreover, we do not allocate any disc space until we really need it. This was, perhaps, a decent solution in the "good old days" when disc space was very expensive. But, now, the operating system would have to maintain 100,001 pointers to enable access to that file, which makes the above method unworkable.

Enter the EXTENT! The extent is a reasonable compromise between the two extreme methods outlined above. A file can consist of anywhere from 1 to 32 extents (the default number is 8). Now, when we build the above file (with 8 extents), we will only have to allocate around 12,500 sectors in the beginning (a savings of disc space) and allocate new extents only every 12,500 records (a savings of disc accesses). We could, however, allocate the file with only 1 extent, thus losing out on disc space but gaining on disc accesses (but, of course, the savings on disc accesses is rather small compared to the incredible wastage of disc space), or with 32 extents,

thus saving disc space at the expense of a few extra disc accesses.

Two other considerations come into play, however — one is that accessing files with a lot of extents FRAGMENTS THE DISC (i.e., increases the number of small holes at the expense of large holes), thus making new files harder to allocate in the future, and another is that it is better to run out of disc space when building a disc file, than when allocating a new extent in the middle of the program (precious time and internal data consistency may be lost this way). The former can be handled best by decreasing the number of extents (at the expense of, of course, disc space) and the latter by allocating at :BUILD-time all of the specified extents (but only if you are sure you will use all of the space). Note that the number of extents (maximum and initially allocated may be specified on the :BUILD command's DEV keyword, whose format is "DEV=device[,maxexts] [,initalloc]", where maxexts defaults to 8 and initalloc to 1.

For Those With Multiple Discs

If you are the proud owner of several disc drives, another factor comes into play. For example, let us say that you build a file with the command ":BUILD ING;DISC=100000" (note that the maximum number of extents defaults to 8, 1 initially allocated), and start to wonder about which disc your file resides on. Well, MPE, has adopted the so-called "eeny, meeny, miney, moe" algorithm. That is, if you succeed in filling all 8 extents of your file, you may well find that that file does not reside on just one disc; rather, it resides on the discs of the DEVICE CLASS "DISC" (which are special sets of different devices, not necessarily discs, configured at system set-up time). Each extent, of course, resides wholly on one disc; but, the extents may reside on different discs — thus, a file with 8 extents may well find itself with 4 extents on disc #1, 2 on disc #2, 1 on disc #3, 1 on disc #4, and 0 on disc #5. If you, however, want that file to reside exclusively on disc #4, "no sweat" (as is said in the vernacular)! Merely :BUILD the file with the "DEV=4" parameter. Or, if you set up another device class called PRODDISC which will contain discs #3, #4, #5, building the file on DEV=PRODDISC will ensure that all extents of that file will be located on one of those devices. What, you may ask, is the importance of this? Well, the word that has leaked down from HP is: SPREAD OUT YOUR FILES — for instance, if you have two heavily accessed files, it might be wise to put them on two different discs.

This is done for the following reason. Let us assume that you have two disc drives, each one able to perform approximately 30 I/Os per second, and you spread out your files in such a way that each disc gets about 30 I/O requests per second. Those requests will be executed within one second. But, if there are 20 I/O requests per second to one disc and 40 to the second disc, the first disc will not perform up to capacity, and 10 of the requests to the second disc will have to wait for a second or more, thus degrading system performance.

Another promising idea is to configure all of your devices except the system disc as device class "DISC," thus keeping files off the system disc, and thus reducing the amount of access to the system disc, which already has the operating system and the virtual memory on it. However, with MPE IV, in which you will be allowed to spread virtual memory over several devices, this may not be as important. Note that for easy file disc location handling, MPEX/3000's %LISTF ,4 and %ALTFILE commands and ADAGER's DBCREATE and SETMOVE functions should be used.

The Logical File Structure

Besides the physical file structure described above — extents, sectors, etc. — MPE files also have an internal logical structure, not enforced in most ways by the actual file contents but rather by certain logical file descriptors like the record size, the blocking factor, the block size, the file type, and the like. First of all, we will discuss the simplest sort of MPE file — the fixed-record length file.

The Fixed Record Length File

A file is more than just a collection of data placed out on disc. It usually has certain logical relationships within it. One of the most frequent and fundamental relationships is one in which data is organized into chunks (called RECORDS) of a fixed length; for instance, if you have a data file which contains, for each customer, the customer code (6 characters), customer name (30 characters), and the amount owed you by the customer (8 zoned decimal characters), you have a 44-character entry for each customer. Therefore, it would be logical, for the sake of ease of access, to build that file with 44-byte (or 22-word) records, having one record per customer. So, to build that file, you would perform a BUILD command with the REC=-44,,F,ASCII parameter (- stands for bytes and F for fixed record length).

The Block

A familiar example of fixed record length disc file is your usual EDITOR /KEEP-NUMBERED file, a file with a record size of 80 bytes = 40 words. However, do you know that in your EDITOR keep files more than 6% of all disc space they occupy is wasted? This may not sound like much, but if you are running short on disc space, this can be a lot. What's more, that disc space can be saved (for large files) by merely specifying a certain :FILE equation for the file to be kept. What, you may ask, is the reason for this wastage? Well, the answer lies in the secrets of the BLOCK.

The fundamental unit of disc I/O (as far as MPE is concerned) is the SECTOR (128 words). Practically all disc I/O ends up as multiples of 128 words. 40, of course, is not a multiple of 128. So, if MPE decided to

place 40 words per sector, it would waste not 6%, but 69% of each sector! So, you ask, why not pack three 40-word records into one 128-word sector. Well, that's exactly what MPE does; but because 128 is not a multiple of 40, either, it still wastes 6% of the file's disc space (although 6% may not sound like much, for some unlucky files which have different record lengths, it can be worse, with up to 50% wasted space!). But, there is light at the end of the tunnel! We can very snugly fit 16 40-word records into 5 128-word sectors — a perfect fit.

From the above labyrinth come the notions of the BLOCKING FACTOR and the BLOCK. The BLOCKING FACTOR is, very simply, the number of records that we choose to fit into a multiple of 128 words — in the above "snug fit" scenario, this is 16; in the 6% wastage method that MPE uses, the blocking factor is 3 (3 records to 1 sector); in the (ugh!) 69% wastage at 1 record to 1 sector, the blocking factor is 1. The BLOCK therefore, is BLOCKING FACTOR records — i.e., when the blocking factor is 16, the block is 16*40 = 640 words = 5 sectors.

In general, MPE chooses the blocking factor as follows. If the record size of a file is less than one sector (128 words), the blocking factor = 128/recordsize = the number of records that will fit into one sector; if the record size of a file is greater than 128 words, the blocking factor is always 1. A good example of the possible wastage is when a record is 65 words long; then, 128/65 = blocking factor of 1, wasting 63 words for every 65 words used — a wastage of 49%! If that record was, however, 64 words long, then the blocking factor would be 2, with NO wastage.

By the way, it happens that the blocking factor for a new file can be defined in a :BUILD or :FILE command — always as the second subparameter (between the record size and the F, V, or U record format) of the REC= keyword. Thus, if you want to eliminate the 6% waste due to the blocking factor of 3 on EDITOR keep files, just execute an equation of the form ":FILE filename;REC=,16" right before keeping the file as "filename," and presto! out comes a file with a blocking factor of 16. For already existing files, some disgustingly complicated tricks can be used — or, if you are blessed with a copy of MPEX/3000, just use the BLKFACT= keyword of the %ALTFILE command.

Now, you may wonder, what leads MPE to choose a default blocking factor calculation system that leads to considerable wastage in perhaps one of the most common forms of files? Well, for one, it would be unfair not to remark at this point that the "NO wastage" schemes described above really DO waste some space (although not a lot). The reason for this is that a file (in fact, each extent of a file) must be an integral number of blocks. If it isn't, a full block is allocated for less than "BLOCKING FACTOR" records. Thus, if you have a file containing 50 80-byte records with a blocking factor of 16, it would use up 4 blocks, the last one having only 2 actual records — this file will thus use 21 sectors; however, if that file is built with a blocking factor of 3, it would use up 17 blocks (the last one also having only 2 records), and would thus use only 18 sectors of disc space. However, this consideration is less important for larger files. Another reason for MPE's default blocking factor strategy is that the block and the blocking factor govern more than just disc space usage — they also control certain parameters of buffered file access (see the chapter on FILE ACCESS). However, for most files (especially large ones!) it is beneficial to select your own blocking factors (with the use of the contributed BLOCK program, for instance).

The Variable Record Length File

Let us take a hypothetical EDITOR COBOL-format file. At the beginning of each line there is a 6-digit line number; the other 74 characters contain the line, blank-padded. Now, those trailing blanks, especially in large source files, convey absolutely no information to anybody, and (since the average length of a line could be estimated at half of 74 characters) will cause a wastage of APPROXIMATELY 50% OF THE DISC SPACE USED BY THOSE FILES! But, you reply, if EDITOR built the file with a record length of, say, 40 characters, all of my lines that are longer than 40 characters will get truncated. Well, you're right — but that is not what is to be done! Wouldn't it be nice if EDITOR and/or the file system allowed you to have files not with a FIXED record length, but with a VARIABLE record length — i.e., lines that are 74 characters long will use 74 characters and lines that are 10 characters long will use 10 characters? Well, it does!

In fact, if you type in the little-known /SET VARIABLE command in EDITOR, it will instruct EDITOR to keep the workfile as a variable length record file (WARNING: USE THIS ONLY FOR COBOL AND DATA FILES, NEVER FOR NUMBERED FORTRAN OR SPL SOURCES, OR THOSE SOURCES WON'T BE COMPILER READABLE!!!), thus letting it ignore those trailing blanks, but still keep the file format transparent to other programs that read these files — for example, compilers. In your own programs (not just in EDITOR), you can read variable record length files without changing your programs at all — COBOL's or FORTRAN's READ command can read variable record length files. You can write them without any changes either — if you write a 10-character record to a fixed record length file of 80 characters, the record will be padded with 70 blanks or nulls; if you write that record to a variable record length file, the record will not be padded by anything, thus saving the space required for the padding. To build a variable record length file, specify the third subparameter of the REC= parameter of the :FILE or :BUILD command as "V" (e.g., REC=-80,,V). The record size specified is now no longer the actual record size of each record but rather the maximum; whether the file is ASCII or BINARY now really doesn't matter. Also, do not call in the Na-

tional Guard (or PICS) when you see on a :LISTF that the END OF FILE for that file is GREATER THAN ITS FILE LIMIT — it can happen with variable record length files.

Therefore, with COBOL source files (especially) and unnumbered data files, variable-length records are usually the way to go; again, however, we must warn you that numbered default-format (e.g., SPL or FORTRAN source) files SHOULD NEVER BE KEPT WITH THE /SET VARIABLE OPTION SET or else they will not be readable by the compiler.

However, as the old proverb says, "EVERY SILVER LINING COMES WITH A CLOUD ATTACHED TO IT," variable record length files have some drawbacks. For one, they can not be accessed directly (for instance, with the FREADDIR, FWRITEDIR, or FPOINT intrinsics, or FORTRAN's READ/WRITE (fnum @ record) construct); i.e., you can read their records sequentially, but you can not ask to get, for instance, the 17th record of the file. Moreover, they cannot be accessed by many file copiers using the fast MR NOBUF file access method (see under FILE ACCESS in this paper), such as HP's own DSCOPY, MPEX's %FCOPY ,,FAST/DSLINE, MPEX's %ALTFILE, SUPRTOOL/ROBELLE, etc. Also, before MPE IV, append access to variable record length files was not supported; it is supported starting with MPE IV.

Another important consideration to keep in mind when using variable record length files is that when you build a new variable record length file with record size RECSIZE and blocking factor BLKFACT, the resultant block size of the file will be not RECSIZE*BLKFACT (as in fixed record length files), but rather RECSIZE*BLKFACT + (BLKFACT+1)*(2 bytes). Thus, if you build a variable record length file of record size 80 bytes and blocking factor 3, the file will actually have a block size of (80*3+4*2)=248 bytes. However, if the same file is built with a blocking factor of 16, the block size will end up being (80*16+17*2)=1314 bytes, not 1280 bytes! The end result is that AN OPTIMAL BLOCKING FACTOR FOR A FIXED RECORD LENGTH FILE MAY BE FAR FROM OPTIMAL FOR VARIABLE RECORD LENGTH FILES!

Incidentally, MPE IV's new INTER-PROCESS COMMUNICATION features (i.e., Message and Circular files) rely EXCLUSIVELY on variable record length files (q.v. COMMUNICATOR issue 26 — the C MIT).

Undefined Record Length Files

There exists another type of disc file — the undefined record length file. These are rather bizarre specimens which are not intended to be and should not be used as disc files, but are rather supposed to be utilized as tape files and terminal files, which are beyond the scope of this paper. ASCII VS. BINARY FILES When using fixed record length files, it often happens that you may write a 30-character record into a file with a record length of 80. Then, what happens to the other 50 characters of the record? Well, for some files (for instance source files) that contain simple text data, you would typically want to initialize it to spaces because of the nature of the file. If that is what you want, you would build that file (EDITOR will build it that way for you) as an ASCII file. This parameter can be specified as the fourth subparameter of the REC= parameter of the :FILE or :BUILD command, e.g., REC=,,,ASCII. However, for some data files, you may want to pad the records with binary zeroes (nulls). Files built in such a way are called BINARY files, and can be built by specifying the BINARY parameter as the fourth subparameter of the REC= parameter of the :FILE or :BUILD command, for example REC=,,,BINARY. Note that this is usually not necessary as BINARY is the default file mode. Also note that since no record padding is done in variable record length files, the ASCII vs. BINARY distinction is usually irrelevant to them.

The File Code

If you do a :LISTF mode 1 or 2 on a group of files, you may notice that some files have a file code of 0 (blanks), some of PROG, USL, EDTCT, KSA M, PRIV, and assorted numeric codes. These filecodes, for the most part, are merely for the sake of file identification — they have no physical influence on the actual contents of files. If you change the filecode of a file (for example with MPEX/30 00's %ALTFILE filename;CODE= command), the contents of the file will not magically change. However, the filecode is useful for identification purposes — for instance, the MPE loader knows that files of filecode PROG are :RUNable program files, the EDITOR knows that files of code EDTCT are /SET FORMAT=COBOL files, QEDIT/ROBELLE knows that files of code 111 are its files. In fact, you can set up your own file identification system for source or data files — you can build files with a certain file code (via the CODE= parameter of a :FILE or :BUILD command), alter the file code (with MPEX/3000 or by copying the file), and examine the file code (via the :LISTF command or, programmatically, with the FGETINFO intrinsic). Certain tools like MPEX also allow you to LISTF files by file code. An example of this kind of file identification system (recently implemented by us) is to set the file code to be the Julian date of the day on which it was created, or some other important date.

Note that the file code of each file is in reality a number — for example, program files (PROG) have a file code of 1029, but they are listed in a :LISTF output as PROG. Also, KSAM files do not actually have a numeric file code that identifies them as such -- they can in reality have any numeric file code. However, KSAM files which have a file code of 0 (which usually shows up

as blanks on a :LISTF listing) will be printed as having code = KSAM. Files that are listed as having file code = PRIV are in reality files that have NEGATIVE file codes (like IMAGE files). Unlike usual files, they can only be accessed by programs running in PRIVILEGED MODE. This is handy, for instance, for IMAGE files, to ensure that an ordinary user can not physically change an IMAGE file without going through the existing IMAGE utilities/intrinsics.

User Labels

It is often desirable or necessary to store information in a file in such a way that it can later be retrieved, but is nonetheless transparent when you read it in an ordinary fashion. The concept of USER LABELS provides this capability. With it, you can write special label records (the maximum number of which is specified at open time, defaults to 0, and can be up to 254) with the FWRITELABEL intrinsic, read them with the FREADLABEL intrinsic, but have them be transparent to any user who reads or writes ordinary records to that file. These labels are used by IMAGE, KSAM, and the message system file (e.g., CATALOG and CICAT). Another advantage of user labels is that you can write user labels when you open the file for read access, can read user labels when you open the file for write access, and can open the file for OUT access (see access modes below) which will else all of the file's records but not its user labels.

Carriage Control Files, Relative I/O Files, Message Files, Circular Files, KSAM Files, IMAGE Files, and Other Monsters That Inhabit the HP3000

This paper will not talk about the above types of files (for want of time, will, and disc space). However, maybe sooner or later you will hear the truth about them, too!

CHAPTER II
FILE ACCESS

Once a file is built, it really isn't much good if you can't access it — read it, write it, append to it, etc. In this chapter we will discuss the different methods of accessing files that MPE provides for you.

Buffered File Access

A while back we referred to the concept of the BLOCK. Well, it turns out that the block is more than a convenient way of storing records on disc. In fact, it plays a very important role in the default mode of file access called BUFFERED FILE ACCESS. Let us assume that you are reading a 10,000-record disc file which has a record size of 40 words (80 bytes), a blocking factor of 16, and thus a block size of 640 words. Let us assume that you had to do one disc I/O for each record — this would come up to a total of 10,000 disc I/O s, quite a lot!

So, MPE implemented a rather ingenious idea called file buffering. Each file opened as a buffered file has allocated for it a certain amount (default 2, changable at open time with the FOPEN intrinsic or the BUF= parameter of a :FILE equation) of buffers, each of length equal to the file's block size (in this case 640 words). These buffers are placed in an Extra Data Segment (because extra data segment access is faster than disc access) and accessed there. They are read from or written to disc only when a record that is not in the buffer is requested. Thus, for the file described above, only 10,000/16 disc I/O's = 625 disc I/O's is necessary — a considerable savings! The advantage of having more than 1 buffer is that then you can access, for instance, records 17-32 (in one buffer) and 49-64 (in the other buffer) without necessitating a disc I/O each time you switch from one record range to the other. However, if you then read in record 100, the contents of buffer 1 will be flushed out to disc and buffer 1 will then contain records 97-112. In general, with buffering, one disc I/O is required for every (BLOCKING FACTOR) records — in this case, one disc I/O is needed for each 16 records.

In the discussion above, we advised that you set up blocking factors so that BLOCKING FACTOR * RECORD SIZE be an even multiple of 128; thus, for instance, 16 was chosen for files with records of length 40 words. But, there is more than one way to skin a blocking factor! In fact, since 16 * 40 is a multiple of 128, 32 * 40 certainly is too! Very little disc wastage will result from changing the blocking factor from 16 to 32, but each buffer will now be not 640 words long, but rather 1280 words long, and now only 313 (=10,000/32) disc I/Os will be necessary to read the file! A blocking factor of 64 will require less than 160 I/O's, and so on. This will not necessarily halve the time used by the read, but it sure will decrease it. Of course, the same thing can be said for writing to files. We must, however, point out that memory space will be used much more heavily by files that have large blocking factors. Also, the total size of the buffers must be less than or equal to 8,192 words (or 14,000 words starting with the D MIT version of MPE). Since the default number of buffers is 2, this puts an upper limit of 4,096 words (or 7,000 words starting with the D MIT) on the block size of a file. However, you can increase that maximum to 8,192 words (or 14,000 words starting with the D MIT) by opening the file with 1 buffer (by specifying BUF=1 on a file equation).

Multiple Record Non-Buffered Access (MR NOBUF)

The buffering method described above is rather good, but is still not optimal; first of all, access to the extra data segment in which the buffers are located is faster than disc access, but nonetheless not as fast as access to

your own stack. Moreover, as was pointed out above, certain memory usage considerations forbid the buffers from being more than 5K to 10K words, which is also not optimal. Wouldn't it be truly wonderful if one could read not just single records, not just blocks of 16 or so records, but 4,000 words at one shot? Well, one can, through the magic of MR NOBUF, probably the MOST POWERFUL AND FASTEST FILE ACCESS METHOD NOW AVAILABLE! MR stands for Multiple Record I/O (do not confuse this with the Multiple RIN capability, also abbreviated MR), and NOBUF stands for No Buffering (this is a bit of a misnomer — it means that it is you, not MPE who will provide the buffer space needed). Note that MR must be used with NOBUF!

Certain factors to beware of when using MR NOBUF are: for one, this method is rather hard to use with variable record length files. Also, the efficiency of this method is best with a buffer size of 4,096 words. Another factor is that when the block size (BLOCKING FACTOR * RECORD SIZE) is not a multiple of 128 words, MR NOBUF is not that much more efficient than ordinary file access, and simple NOBUF access should be used instead.

By far, the best application of MR NOBUF is with file copying. FCOPY, which uses ordinary buffering methods, is often 10 to 20 times slower than MR NOBUF copiers like HP's own DSCOPY, MPEX/3000s %FCOPY ,,FAST/DSLINE or %ALTFILE commands, SUPRTOOL/ROBELLE, and numerous other programs. However, you can do MR NOBUF reading and writing from your own programs by specifying the MR NOBUF access options when accessing the file (or specifying the MR or NOBUF parameters on the :FILE equation — however, a bug present on some versions of MPE forces you to specify MR in the FOPEN because it ignores the MR :FILE equation parameter; see "ANOTHER MPE FEATURE (BUG)" in SCRUGletter, Jan 1981 Vol 4 #1). This will allow you to read more than one record (always at least one block, however) at a time, and also lets you do direct I/O (e.g., with FREADDIR and/or FWRI TEDIR) on a block number rather than record number basis.

However, reading files MR NOBUF in your own programs is rather hard to do because of many concerns that have to do with doing deblocking of records. Because of this, it is suggested that you either do most of your record selection outside of your program (with SUPRTOOL/ROBELLE, for instance), develop your own MR NOBUF I/O routines that can be easily called from your applications programs, or use David Brown's FAST I/O procedures.

MR NOBUF I/O can therefore really cut down the execution time and CPU time demands of your disc I/O-heavy programs. The only problems with MR NOBUF are that it is hard to apply it to variable record length files and KSAM files and that it may (because of the large in-stack buffers necessary) use up a lot of stack space and much memory space.

The Access Types for Disc Files

In the access options parameter of the FOPEN intrinsic or in the ACC= parameter of the :FILE command you can specify the so-called access type which defines whether a program will read the file, write to the file, do both, or append records to the file. There are 7 legal access types, which can be very useful if used properly.

The default access type is IN access. This is read-only access — all attempts at writing records to files opened with IN access will fail with File System Error 40 — OPERATION INCONSISTENT WITH ACCESS TYPE. If you only want to read the file, you should open it with this access type; this will prevent your program accidentally writing over the file; it will work even if somebody else has the file opened in Share or Exclusive Allow Read mode (see the SHARING FILES chapter) and it will also work if the file's security prevent you from doing anything but reading that file.

Another type of access is OUT access. OPENING OLD FILES WITH THIS ACCESS TYPE WILL ERASE THEM! If you do not want that to happen, you should open th e file with OUTKEEP access. However, if you want to erase the file, or the file is new, or you do not care about its old contents anyways, this is the access type that should be specified. Note that you need WRITE access to the file to open it in this mode. OUTKEEP access is useful for opening files to write to them, but NOT DESTROYING THEIR OLD CONTENTS (as OUT access would do). You need WRITE access to the file to open it with this access type.

Often you do not need to write over the old contents of a file — you merely need to add new records to it. In that case, APPEND access is for you — it forces the record pointer to be positioned at the end of the file and only permits you to append records to the file. Another advantage of it is that it requires that you have only APPEND access (not WRITE access) to the file to open it thus. So, if you wish to permit users to only append and not overwrite data in a given file, they should be allowed only APPEND and not WRITE access to this file. For instance, VESOFT's SECURITY/3000 permits APPEND access to its security violation log file, but not WRITE access (so user's can not obliterate the record of their violations).

The above access types permit you either to READ ONLY or WRITE ONLY, but never both. INOUT access lets you both READ and WRITE to the file. All intrinsics (except FUPDATE) can be used against that file in this mode. Note that you need READ and WRITE access to open a file in this way.

There is also a special form of access called UPDATE access that is PRECISELY the same as INOUT access except that it permits the usage of the FUPDATE intrinsic. Since this is apparently no less expensive than INOUT access, and requires no extra access to the file,

it is suggested that this option be used instead of the INOUT access type because it is more powerful and no more dangerous.

Another access, permissible only to programs that run in Privileged Mode (Ohmigod!), is EXECUTE access; its advantages are twofold. For one, it requires only EXECUTE access to a file, not READ access; moreover, it allows you to write to loaded program or SL files. This is listed only for the sake of completeness, and all you nice non-privileged users out there don't even need to know about it. For a discussion of privileged mode, see PRIVILEGED MODE: USE VS. ABUSE, SCRUGletter July 1981, Vol 4, #4.

Posting End of File to Disc

As was mentioned before, each file has a special record called "the file label" (which contains all sorts of information about the file, such as its type, name, and, among other things, its end of file, which is the number of records which the file contains). Now, if every time that you wrote a record to the file, MPE would have updated that file's file label, your programs would run quite slow — after all, that would mean extra disc I/Os to handle. For this reason, MPE does not post the end of file to disc until a record write would cause it to allocate a new extent (in which case it would have to change the file label anyway), thus saving the extra I/Os.

This is all fine and dandy, provided that MPE will actually get a chance to post the end of file to disc sometime. But, what if the system crashes after you wrote the record but before MPE posted the end of file? Then, even though the record (or records, as the case may be), are already out on disc, MPE does not know about it because the end of file pointer does not reflect this. So, you've just lost all those records that were written before the system crashed. You can, however, minimize your losses through a little-known feature of the file system by calling the FCONTROL intrinsic (see System Intrinsics Manual) with a parameter 6 (WRITE END OF FILE) which lets you post the end of file to disc. If you do this after you write each record, the most records that you will ever lose due to a system failure is one! Of course, this will triple the number of disc I/Os that you'll have to do, so this is not advised for large batch runs; however, if you are updating a disc file interactively, the time it takes to input all of the data from the screen will dwarf the time it will take to do the extra I/O to such a degree that that the posting of the end of file will be virtually free in terms of time, and may save you hours of re-entering vital data.

When You Are Not Alone

When you use any of the access types listed above except read only (IN) access, the file specified will be opened EXCLUSIVELY; that is, you can not open it if anybody else has it opened, but, once you have it opened, NOBODY ELSE CAN USE IT UNTIL YOU CLOSE IT. This is, of course, somewhat of a problem if that file is intended to be read and written by many different users. There are several ways to get around this dilemma.

Exclusive Allow Read Access

One of those ways is Exclusive Allow Read (EAR) access. This permits you to forbid all other users from writing to a file, while letting them read that file. Also, this access (unlike EXCLUSIVE) access will be granted to you even if the file is already being accessed for read access (but not for write access) by someone else. This can be specified by setting the appropriate bits in the access options of the FOPEN call, or issuing a FILE equation with the EAR keyword. TRUE SHARED ACCESS But, you sometimes want not just to have one writer of a file, or one writer and several readers, but MORE THAN ONE PERSON WRITING TO A GIVEN FILE. This can be accomplished with SHARED ACCESS (to use, specify the appropriate bits in the FOPEN call or append the SHR keyword to the file equation for that file), which is the default mode for read only access, but has to be explicitly specified and handled when writing to a file. Shared access is a very complicated form of access, one at which we will look closer in the next chapter.

CHAPTER III
SHARED FILES

Sharing Files With Input/Output Access

Merely specifying SHR access when opening the file will get you where you are going — it will allow you and anybody else who opens the file with SHR access to read and write to this file. But, let us suppose the following situation: two processes have opened one file for IN/OUT access in SHR mode, and the following happens:

PROCESS A	PROCESS B
Reads a record	
	Reads the same record
Changes the record	
	Changes the record
Writes the record back	
	Writes the record back

In the above scenario, process A reads the record before process B reads the same record but writes it back out after process B reads it in! That way, process A's changes WILL NOT BE REFLECTED IN THE FILE because of the interference of process B. In fact, what is needed is a method of "LOCKING OUT" all other writers of the file while the file is being updated! Well, MPE's FLOCK and FUNLOCK intrinsics provide this method.

Dynamic Locking and Unlocking for Shared Files

In order to use dynamic locking, the process that opens the file must open it with dynamic locking enabled (the LOCK parameter on the :FILE equation — together with the IN/OUT SHR access, the file equation would now look like :FILE file;LOCK;SHR;ACC= INOUT — or the appropriate bit in the access options of the FOPEN intrinsic call). Then, before each "logical transaction" (a period in time in which the data in the file is not consistent — in the above example, while the record is being changed, the current state of the file does not reflect the true intended state; therefore, the file must be locked before the read and unlocked after the write) the file must be locked and then be unlocked after the end of the transaction (note that opening a file with dynamic locking enabled does not actually lock the file). This will ensure that there will be no inconsistencies like the one shown above. Note that THIS WILL WORK ONLY IF ALL WRITERS LOCK THE FILE IN APPROPRIATE CASES — this locking arrangement only works for programs that honor it.

Sharing Files with Append Access

In some cases, however, locking does not really help. For example, if two writers are just writing to a file (no reading, etc.), the "logical transactions" like the ones described above are composed of merely one write. For these transactions, it does no good to lock the file. One of the most common example of this type of file access is shared append access to a file by two or more writers.

In fact, if the file has a blocking factor of 1, there is no need to do anything but the write. However, look at what happens when the file has a blocking factor other than 1, for example 3; consider process A and process B, both writing to the same file:

PROCESS A	PROCESS B
Record 1 written; kept in buffer	
	Record 1 written; kept in buffer
Record 2 written; kept in buffer	
	Record 2 written; kept in buffer
Record 3 written; buffer flushed	
	Record 3 written; buffer flushed

Note that, by the principles of buffering, the actual disc I/O is not done until the third record is written and the buffer is flushed out to disc. But, because of that, when it is flushed out to disc, the buffers from process A and process B interfere with each other, and data can be lost. Therefore, the rule for locking when appending (or performing any other such operation in which each transaction contains only one operation) is: LOCK WHEN THE BLOCKING FACTOR IS GREATER THAN 1; IF THE BLOCKING FACTOR IS 1, LOCKING I S UNNECESSARY.

Multiple File Access

Another way to ensure that no data is lost while writing to a file is with a useful tool (which is even more useful under MPE IV) called MULTIPLE FILE ACCESS. With multiple file access in shared mode, the internal file control information and the I/O buffers are shared, as well as the file itself, thus avoiding many problems of ordinary shared access.

So, if process A and process B (IN THE SAME JOB/SESSION) access a file SHARED, APPEND, and MULTI, then their internal end of file and buffer pointers are shared; thus, the risk of one's file I/O interfering with the other's is eliminated. To specify MULTI-access, set the appropriate bit in the FOPEN parameters or specify the ;MULTI keyword on a :FILE command for the file in question.

So, very many of the problems and complicated locking strategies discussed above can be avoided if MULTI-access to that file is used. However, there are two things that you must keep in mind when using MULTI-access; for one, ordinary sequential reads and writes to that file will not behave as expected. Why? Well, the current record pointer is among those values that is shared with MULTI-access and thus if process A reads a record sequentially and then process B requests to read a record sequentially, process B will get the next record because the record pointer was already incremented by A's read. Thus, if the two processes read the file sequentially with MULTI-access, each one will read approximately half the file instead of the full file!

MPE III vs. MPE IV

Another problem for all you unlucky people who still do not have MPE IV, MULTI-ACCESS IS PERMISSIBLE ONLY WITHIN ONE JOB/SESSION UNDER MPE III! However, under MPE IV, you can use the GMULTI (Global MULTI access), which can be specified in the FOPEN parameters or with the GMULTI keyword of the :FILE equation, to have MULTI-ACCESS ACROSS JOBS/SESSIONS, with which you can avoid most of the problems of shared file access very easily.

More About Locking

There are two methods of locking files: UNCONDITIONAL, which means "if the file is already locked by somebody else, wait for them to unlock it, and then establish the lock" and CONDITIONAL, which means "if the file is already locked by somebody else, return to me immediately with an error condition." The UNCONDITIONAL method is usually the most useful, although the CONDITIONAL option is handy when you

do not want to take the risk of waiting a long time (if the program that has it locked won't unlock it for a while). Needless to say, the file should not be locked for a long time, and SHOULD NEVER BE LOCKED WHILE A TERMINAL READ IS GOING ON unless you do not mind the fact that if the terminal operator goes to lunch, everybody else who tries to unconditionally lock the file will hang.

Locking Multiple Files, Or The Secrets of Multiple Rins (MR)

Let us consider another hypothetical circumstance: Process A locks File 1; meanwhile Process B locks File 2. Then, Process A tries to unconditionally lock File 2 and is then impeded until Process B unlocks File 2. Meanwhile, Process B tries to unconditionally lock File 1 and is then impeded until Process A unlocks File 1. Thus, Process A is waiting for Process B and Process B is waiting for Process A. Result: Deadlock. Both processes are hung until the system is re-started. The sages of Cupertino thought of that when designing the system; in fact, their solution (which may not sound like much of a solution, but is better than nothing) is TO FORBID PROGRAMS TO LOCK MORE THAN ONE FILE AT A TIME. But, one may object, what if I have to lock more than one file at a time? Well, the answer to that problem is that you can get around (but at your own risk) that restriction if the program that does the locking has Multiple RINs (MR — not to be confused with Multiple Record access) capability (i.e., was :PREPped with it. By the way, RIN stands for Resource Identification Number. These programs can, IF THEY REALLY HAVE TO, lock two or more files at a single time. Needless to say, this capability should not be freely given to everybody and his brother, but only to people who really need it, and smart enough to use it without causing deadlocks.

That brings us to the problem of: How do you get around the deadlock problem? Well, you may have already noted that the reason why the programs got into a deadlock was that one locked File 1 before File 2 and the other locked File 2 before File 1. If they had only kept a consistent locking arrangement (e.g., File 1 must ALWAYS be locked before File 2), they would not have had the problem — this is probably the best way to avoid the deadlocks. Another way is to lock the files CONDITIONALLY, and if the lock fails, do something else (or even go into a loop, which can at least be broken out of by aborting the job or doing a break/:ABORT, rather than re-starting the system).

Summary of Locking And Locking Strategy

The following are the 10 commandments of locking:

1. *Thou shalt lock around logical transactions which involve two or more operations.* For example, that kind of a logical transaction would be a read of a record followed by a modification of that record followed by a write. If you do not lock around this, you stand the risk of losing data consistency.

2. *Thou shalt also lock around all logical transactions that involve a file which you share with somebody who has transactions which involve two or more operations.* That means that if process A's transactions are just single writes and process B's transactions are reads followed by writes, both process A AND process B must lock around their transactions.

3. *Thou need not lock a shared file if all its writers' transactions involve just one operation and its blocking factor is 1.* Thus, if process A and process B are writing to a shared file, and their transactions are merely single writes (e.g., they are appending to a file), neither one has to lock the file.

4. *Thou shalt use GMULTI access under MPE IV when you are appending to a shared file.* This can save you time, worry, and your neck.

5. *Honor thy locking arrangements.* This means that if it has been decided that a shared file is to be locked by its writers, all writers must lock it. If so much as one writer fails to lock the file, all of the locking arrangements will be useless.

6. *Thou shalt not keep a file locked while a terminal read is in progress.* If you did, then the file will be locked down until something is entered, which could mean an indefinite waiting period for any other program that wants to lock the file.

7. *Thou shalt not lock more than one file at the same time without MR Capability.* The second file lock will fail unless your program was :PREPped with MR capability.

8. *Thou shalt protect thyself from deadlocks by establishing a fixed file locking sequence if you use MR capability.* Thus, if process A locks file 1 and then file 2, process B must lock in the same order, i.e., file 1 and then file 2 (not file 2 and then file 1!).

9. *Thou shalt not give MR capability to just anybody.* MR capability can cause big trouble, and thus should be passed out sparingly.

10. *Thou shalt use IMAGE/3000 if thy file locking arrangements get too complicated.* IMAGE/3000 has file locking capabilities far superior to MPE's file locking features. If you find that your locking arrangements are getting too complicated or programs are waiting inordinate amounts of time to get at a shared file, think about converting it to an IMAGE file — it may be worth your while.

CHAPTER IV
FILE DOMAINS AND EQUATIONS

Permanent and Temporary Files

Most of the files that we discussed in previous sections were usual PERMANENT files — files that, once built, exist until they are :PURGEd or somehow deleted. There is, however, another type of file, one that is

also often quite useful. This is the JOB/SESSION TEMPORARY FILES. These files, once built (by placing the ;TEMP keyword on the :BUILD or :FILE command), exist until they are :PURGEd (by performing a ":PURGE filename,TEMP") OR UNTIL THE JOB OR SESSION IN WHICH THEY WERE CREATED LOGS OFF. Why are these files desirable? Imagine, for instance, that you want to create a certain file that you want to stream. After the file is streamed (in the same job or session that it was built in), you no longer need it. If you were to create that file as a permanent file and then purge it, it is quite possible that somebody else may have built a file with the same name; for instance, if the same program is being run on another terminal and that file is created there.

However, if you create it as a temporary file, you can be certain that creating it will not interfere with anybody else; the nature of job/session temporary files is such that two different jobs or sessions can create within them temporary files with the same name which do not interfere with each other.

Most MPE commands either attempt to open the file given to them as a temporary file and then (if the temporary open fails) as a permanent file (e.g., :STREAM,:COBOL,:RUN, etc.), thus being able to accept both temporary and permanent files, or have special keywords that instruct them to open the file as a temporary file (e.g., PURGE file,TEMP). Programs that open files as permanent can be instructed to open the file as job/session temporary by issuing a file equation of the form ":FILE fil ename,OLDTEMP". Note that some commands and subsystems (e.g., :BASICOMP, :PREP, :SEGMENTER's -BUILDUSL command) build files as temporary files; others can be instructed to build files as temporary by using a file equation like ":FILE filename;TEMP".

If you need to keep a temporary file as a permanent file with the same name, you can do a ":SAVE fil ename"; if you want to keep it as one with a different name, do a ":RENAME oldfile,newfile,TEMP" and a ":SAVE newfile". The names of your temporary files can be listed with LISTEQ2 or (in a more complete, :LISTF-like format, with MPEX/3000's %LISTF fileset: TEMP command).

$NEWPASS and $OLDPASS

Two other useful critters are the system-defined files called $NEWPASS and $OLDPASS. Consider, for instance, the :COBOL command. When the USL file is omitted on this command, it is usually followed by a :PREP command that is to prepare the resultant USL file into a program file. But, what intermediate USL file should be used? Well, if you use a permanent or temporary file you run the risk of having a file with that name already in existence. This is where $NEWPASS and $OLDPASS come in. $NEWPASS is a peculiar file that, when closed, magically turns into $OLDPASS. So, once you open $NEW PASS, write to it, and close it, you can then open $OLDPASS, and read it.

So, in the case of the :COBOL and :PREP, the USL file parameter of the :COBOL command defaults to $NEWPASS. The USL file is closed, and, presto!, it becomes $OLDPASS. Now, you can execute a command of the form ":PREP $OLDPASS,progfile", and that USL will be :PREPed into the specified program file. If you really want to be fancy and you don't need the program file to be a temporary or permanent file, you can do a ":PREP $OLDPASS,$NEWPASS", and, after this is done, the program file (which was specified as $NEWPASS) becomes $OLDPASS. Now, you can just ":RUN $OLDPASS". Note that $OLDPASS contains the USL file from :COBOLPREP (or :FORTPREP, :SPLPREP , etc.) and the program file from :COBOLGO (or :FORTGO, :SPLGO, etc.).

If you decide that you want to save the contents of $OLDPASS in a permanent file, just do a ":SAVE $OLDPASS,filename". A rather bizarre undocumented feature is that to save $OLDPASS as a TEMPORARY file, you can do a ":RENAME $OLDPASS,filename"! Of course, $OLDPASS vanishes as soon as you :BYE off.

The Care and Feeding of :File Equations

Perhaps one of the single most important and least understood tools in handling files is the :FILE equation. The file equation allows one to re-define certain open parameters of old and new files. For example, let us say that you are keeping a file with EDITOR, and you want to keep it with blocking factor 16 and 32 extents. Then, you would issue the file equation ":FILE filename;REC=,16;DEV=,32". Note that THIS DOES NOT BUILD THE FILE! However, when you execute the /KEEP command (and EDITOR therefore opens the file) or when you open it from your own or any other program as a new file, it will be opened with blocking factor 16 and 32 extents.

If, however, the specified file already exists and has a blocking factor of 3 and 8 extents and you issue the file equation in hopes that the equation will magically transform it, you're in for a letdown. This is because if that file already has a blocking factor of 3, it will always have a blocking factor of 3 even if you say on the :FILE equation or when opening the file that it has a blocking factor of 16. Its blocking factor is 3 and merely opening it with another blocking factor changes nothing. To truly change the blocking factor, record size, number of extents, file limit, or any one of the other file parameters, you need to either rebuild the file (remember, these parameters can be redefined when you are building a new file) and copy the old contents of the file into it, or use utilities such as MPEX/3000.

However, some options can be redefined for OLD files. These are not the file options (like CCTL or REC) but the access options (like ACC, BUF, MR, etc.), which are not inherent parts of the file, but rather at-

tributes of the access, defined when the file is opened. These can therefore be redefined for OLD or NEW files. Another class of :FILE equation parameters governs actions that are to be performed not at OPEN time, but rather at CLOSE time. The only parameters in this class are disposition parameters. The SAVE option instructs the program to close the file as a permanent file; the TEMP option tells it to close the file as a job/session temporary file (q.v. TEMPORARY vs. PERMANENT FILES); and, the DEL option will delete the file referenced when it is closed. Note that although all :FILE equation parameters correspond to some FOPEN or FCLOSE parameter, not all FOPEN and FCLOSE parameters can be redefined with a :FILE equation; for instance, the number of user labels (on open), or the flag that indicates whether space between end of file and file limit is to be released (on close) can not be redefined with :FILE equations.

If you do not want the user to be able to re-define the open or close parameters of a file, you should open the file with the Disallow File Equations bit in the FOPTIONS parameter of the FOPEN intrinsic set.

APPENDIX B

A Glossary of Common Disc File Handling Terms

ACCESS-MODES — The file's ACCESS MODE (one of IN, OUT, OUTKEEP, APPEND, INOUT, UPDATE, or execute) that is defined at file open time and restricts the actions that can be performed on the file. This can be redefined with the ACC= parameter of the :FILE equation. See APPEND ACCESS, IN ACCESS, INOUT ACCESS, OUT ACCESS, OUTKEEP ACCESS, UPDATE ACCESS.

ACCESS-OPTION — The ACCESS OPTIONS are a parameter to the FOPEN intrinsic (q.v.) that define the access mode, sharing status, dynamic locking flags, etc. See FOPEN.

ASCII — ASCII files are fixed/undefined length files that are padded or initialized to blanks instead of zeroes. That is, writing a record that is shorter than the record size causes the result to be blank-padded. To create, use ASCII as the 4th subparameter of the REC= parameter of the :FILE/:BUILD command. See BINARY.

BINARY — BINARY files are fixed/undefined length files that are padded or initialized to zeroes (nulls). To create, use BINARY as the 4th subparameter of the REC= parameter of the :FILE/:BUILD command. See ASCII.

BLOCK — A BLOCK is the unit in which data is transferred between I/O devices and file buffers on disk. 1 BLOCK = BLOCKIN G FACTOR records. Each block always starts on a sector boundary, and thus, for disc space usage efficiency should be equal to an integral number of sectors whenever possible. See BLOCKING FACTOR, BLOCK SIZE.

BLOCKING-FACT — The BLOCKING FACTOR is the number of records per block. To optimizing disc space usage, set the blocking factor such that BLOCKING FACTOR * RECORD SIZE is a multiple of 128 words. To optimize file access speed, set the blocking factor as large as possible. To minimize memory usage, set the blocking factor as small as possible. To set the BLOCKING FACTOR for new files, specify it as the 2nd subparameter of the REC= parameter of the :FILE or :BUILD command. See BLOCK, BLOCK SIZE.

BLOCK-SIZE — For fixed record length files, BLOCK SIZE = BLOCK FACTOR * RECORD SIZE. For variable record length files, BLOCK SIZE = BLOCK FACTOR * RECORD SIZE + (BLOCK FACTOR + 1) * (2 bytes). The most efficient disc space usage occurs when the block size of a file is equal to an integral number of SECTORS. See BLOCK, BLOCK FACTOR.

BUFFERING — The default mode of file access is BUFFERED FILE ACCESS — in this mode records are not immediately read from or written to disc, but rather kept in an extra data segment which contains (BUFFERS) buffers of length (BLOCK SIZE) words each. See BUFFERS, NOBUF ACCESS.

BUFFERS — When a file is accessed in buffering mode, a certain number of BUFFERS is allocated, each one of length (BLOCK SIZE) words, in one extra data segment. The default number of buffers is 2, and can be redefined with the BUF= parameter of a file equation. See BUFFERING.

DEADLOCK — A situation in which two processes are hung, each one waiting for the other to do something. This can happen when several files are locked by processes with MR capability. See LOCKING FILES, MR CAPABILITY.

DEVICE — The DEVICE on which a disc file resides can be a single disc (specified by placing its device number in the FOPEN call or as the 1st subparameter of the DEV= keyword of the :FILE equation) or a device class, a collection of disc devices grouped under a generic name (specified in the same place as the device

number). All of the extents of the file are placed on this device or device class.

DOMAIN — The DOMAIN of a file can be PERMANENT or TEMPORARY. This can be specified on a :BUILD command (;TEMP indicates TEMPORARY, omission of it means PERMANENT) or a :FILE command (for old files, :FILE filename,OLD means PERMANENT and :FILE filename,OL DTEMP means TEMPORARY; for new files, :FILE filename;TEMP means TEMPORARY and ;SAVE means permanent). See PERMANENT, TEMPORARY.

EAR — EAR (short for Exclusive Allow Read) is an access mode that permits a user to open a file for write access, but still allow other users read access to the file. See EXCLUSIVE ACCESS, SHARED ACCESS.

END-OF-FILE — The END OF FILE is usually the number of records that have been written to a given file. It is usually less than the file limit (q.v.), which is the maximum number of records in a file, but could be greater than it in variable record length files (q.v.).

EXCLUSIVE — EXCLUSIVE ACCESS to a file is an access mode in which the accessor forbids everybody else to access that file while he is accessing it. This mode is the default mode for all non-read access. It can be specified in the access options of an FOPEN call or in a :FILE equation with the EXC parameter. See EAR ACCESS, LOCKING, SHARED ACCESS.

EXTENT — An EXTENT is a collection of blocks that occupies contiguous space on a given disk. There can be up to 32 such extents in a file, but the default is 8. See EXTENT SIZE, MAXIMUM EXTENTS, NUMBER OF EXTENTS.

EXTENT-SIZE — The extents of any file must all be of equal length (in sectors), except the last one, which may be of smaller length. For formulae for these lengths, see APPENDIX B — DETERMINING DISC SPACE USAGE. See EXTENT.

FILE-CODE — The FILE CODE of a file is an integer which describes the type of this file; some of the more common codes have special mnemonics corresponding to them (e.g., PROG = 1029 = file code of program files). These mnemonics show up on :LISTFs of that file, and can also be specified on a :BUILD or :FILE command. The code, whether mnemonic or numeric, can be placed on the CO DE= parameter of a :BUILD or :FILE command.

FILE-EQUATION — A file equation is a useful tool that allows a user to redefine certain open or close parameters of the file (e.g., the file code (CODE), the access type (ACC), the close disposition (SAVE/TEMP), etc.). It can be specified through the MPE :FILE command.

FILE-LABEL — The FILE LABEL of a file contains information about that file (e.g., file name, creator id, file code, record size, extent information, etc.) needed by MPE. Ordinary users need not worry about this entity.

FILE-LIMIT — The maximum number of records permitted in a file, necessary for knowing how much disc space to allocate, specified at file creation time. Note that the END OF FILE can actually exceed the FILE LIMIT for variable record length files. The file limit can be specified in a :BUILD or :FILE command as the first subparameter of the DISC= keyword.

FIXED-LENGTH — FIXED RECORD LENGTH files are files whose records have a fixed length — if a record of smaller length is written to the file, the record is padded on the right with an appropriate number of blanks (ASCII files) or nulls (BINARY files). An example of this kind of file is the usual EDITOR file which has a fixed length of 80 bytes. To build fixed record length files, specify F as the third subparameter of the REC= parameter on a :BUILD or :FILE command (e.g., REC=-80,,F). See VARIABLE RECORD LENGTH, UNDEFINED RECORD LENGTH.

FOPEN — FOPEN is a system intrinsic that permits its caller to open a file. BASIC, COBOL, FORTRAN, and RPG users need not be concerned about this intrinsic because their languages provide file access features already (this is therefore mostly used by SPL programmers); however, we often allude to this intrinsic in this paper because all file open commands in all languages eventually translate out to this intrinsic.

GMULTI-ACCESS — GMULTI access is an extended form of MULTI access (q.v.) available only on MPE IV. Its usage (which can be specified by appending the GMULTI keyword to the :FILE equation) together with SHR and ACC=APPEND provides a painless way of appending to shared files. See MULTI, SHARED ACCESS.

LOCKING — MPE's DYNAMIC FILE LOCKING mechanism (available through the FLOCK and FUNLOCK intrinsics) gives users a way to have more than one program write to a file without jeopardizing data consistency. In order to call FLOCK and FUNLOCK, the file must have been previously opened with the dynamic locking access option set (which can be done in the FOPEN call or using the LOCK parameter of the :FILE command). See DEADLOCKS, MULTIPLE RINS, SHARED ACCESS.

MAX-EXTENTS — The MAXIMUM NUMBER OF EXTEN TS defines into how many extents

(q.v.) a file is to be split. Note that (usually) not all of these extents are allocated at the time a file is built — the default is 1 (although more can be allocated initially by specifying their number as the 3rd subparameter of the DISC= keyword of the :FILE command). The maximum number of extents can be specified as the 2nd subparameter of the DISC= keyword of the :FILE command, and defaults to 8. See EXTENTS, NUM EXTENTS.

MULTI-ACCESS — MULTI access is a form of access that is very useful for sharing files. It permits you to share not just the files but also internal file control information and file buffers. It can be specified by placing the MULTI keyword on a :FILE command. See GMULTI ACCESS, SHARED ACCESS.

MULTIPLE-RINS — The MR (MULTIPLE RINS) capability is an account, file, group, and user capability that governs a program's ability to have more than one file locked at at a time. In order for a program to be permitted to do this, it must have been :PREPped with MR capability by a user who had MR capability, and it must reside in a group that has MR capability. See DEADLOCKS, LOCKING.

MULTI-RECORD — MULTI-RECORD ACCESS (abbreviated MR) is a mode in which a file accessor can read more than one record at a time, thus greatly speeding up file access. This option must be used together with the NOBUF option (see NOBUF ACCESS). It can be specified on a :FILE equation as the MR parameter. See NOBUF ACCESS.

$NEWPASS — $NEWPASS is a special system-defined temporary file that, when closed, turns into $OLDPASS (q.v.). This file (and $OLDPASS) disappear (along with all job/ session temporary files) at logoff time. See $OLDPASS, TEMPORARY FILES.

NUM-BUFFERS — The NUMBER OF BUFFERS is the number of I/O buffers allocated for buffered file access (q.v.). This number can be specified with the BUF= parameter of a :FILE equation. See BUFFERING.

NUM-EXTENTS — The NUMBER OF EXTENTS is the number of extents that that are currently allocated in the file; this starts out as the initially allocated number of extents (see EXTENTS), and is increased by 1 whenever a record is written to the file which will not fit into the currently allocated number of extents. See EXTENTS, MAXIMUM EXTENTS.

$OLDPASS — $OLDPASS is a special system-defined temporary file that was the last $NEWPASS (q.v.) file closed. This file disappears at logoff time, but can be saved with the MPE :SAVE command. See $NEWPASS, TEMPORARY FILES.

PERMANENT-FILE — A permanent file is a disc file that is accessible by all users in the system (that have the proper access to it, of course) and remains until it is :PURGEd, as opposed to a temporary file (q.v.) that can be accessed only by the session in which it was created and is automatically deleted when that session logs off. The fact that a file is to be accessed as an OLD permanent file can be specified by executing a file equation of the form ":FILE filename,OLD"; the fact that a file is to be saved as a NEW permanent file can be specified by placing the SAVE keyword on a :FILE equation for that file. See TEMPORARY FILES.

RECORD-LENGTH — The RECORD LENGTH of a file is the length of each records in that file if it is a fixed or undefined record length file, and the maximum length of the records in that file if it is a variable record length file. This parameter can be specified as the 1 st subparameter of the REC= parameter on a file equation. See FIXED RECORD LENGTH, UNDEFINED RECORD LENGTH, VARIABLE RECORD LENGTH.

SECTOR — A SECTOR is 128 words of disc space.

SHARED-ACCESS — Files open in SHARED ACCESS mode can be written by more than one program at the same time. This option can be specified in a :FILE equation with the SHR parameter. It is imperative for data consistency that the dynamic locking (q.v.) facility be used by all programs that write to a file shared by two or more writing programs. See EAR ACCESS , EXCLUSIVE ACCESS.

TEMPORARY-FILE — A temporary file is a disc file that can be accessed only by the session that created it, and is automatically purged when that session logs off. It can, however, be saved as a permanent file (q.v.) with the MPE :SAVE command, and purged before the session logs off with the PURGE filename,TEMP command. The fact that a file is an OLD temporary file can be specified by using a file equation like ":FILE filename,OLDTEMP"; the fact that it is to be saved as a NEW temporary file can be specified by appending the TEMP keyword to a :FILE equation for that file. See PERMANENT FILES.

UNDEFINED-LEN — Undefined record length files are not intended to be used as disc files. Use instead fixed / variable record length files. See FIXED RECORD LENGTH, VARIABLE RECORD LENGTH.

USER-LABELS — User labels are records which, al-

though they are parts of the file, are transparent to the normal reader of that file, and can only be accessed via the FREADLABEL and FWRITELABEL intrinsics.

VARIABLE-LEN — Variable record length files are files in which not all records have to have the same length. When records of length less than the record length (which, incidentally, is the maximum length of any record in that file) are written to the file, no padding is done (which means that the ASCII / BINARY distinction has no meaning here), but rather the size of the record to be written becomes the record length of that particular record. A result of this is that no space is wasted due to padding, which makes these files much more efficient users of disc space than fixed record length files (q.v.). See FIXED RECORD LENGTH, UNDEFINED RECORD LENGTH.

APPENDIX B

Determining Disc Space Used By Files Given File Parameters

Perhaps because there are so many different file parameters (record size, blocking factor, end of file, file limit, etc.) that are involved in determined the disc space used up by a certain file, the formula for this calculation is hard to come by and is quite complicated. However, we will attempt to list it together with all its interesting ramifications below. Note that this method will work only for FIXED RECORD LENGTH FILES that are to be WRITTEN IN A SEQUENTIAL FASHION (i.e., no directed writes). The parameters needed for this algorithm are the RECORD SIZE (in words), BLOCKING FACTOR, END OF FILE, FILE LIMIT, NUMBER OF USER LABELS, and MAXIMUM NUMBER OF EXTENTS REQUESTED. This method will yield the NUMBER OF SECTORS USED BY THE FILE, THE EXTENT SIZE OF MOST EXTENTS, THE EXTENT SIZE OF THE LAST EXTENT OF THE FILE, THE MAXIMUM NUMBER OF EXTENTS GRANTED, THE NUMBER OF EXTENTS ACTUALLY USED, and THE BLOCK SIZE OF THE FILE.

Blocking Considerations

The first parameter that must be determined for this calculation is the BLOCK SIZE, in SECTORS, which we will denote by the "variable" name BLKSIZE. Using standard SPL notation, the names BLKFACT = blocking factor and RECSIZE = record size, and keeping in mind that ALL DIVIDES PERFORMED BY US FROM NOW ON WILL BE "CEILING" DIVIDES, i.e., DIVIDES IN WHICH THE RESULT IS THE SMALLEST INTEGER THAT IS LARGER THAN OR EQUAL TO THE FRACTIONAL DIVIDE RESULT (e.g., 5/2=3, 20/4=5), we get the following formula:

```
BLKSIZE:=(RECSIZE*BLKFACT)/128;    << Record size IN WORDS >>
```

Next, we must find out the number of blocks (not records, but blocks) that are used up by the data portion of the file and the label (user label and file label) portion of the file. The formulae for this (note FLIMIT = file limit, ULAB = number of user labels allocated) are:

```
DATABLKS:=FLIMIT/BLKFACT;
LABBLKS:=(ULAB+1)/BLKSIZE;   << the 1 is for the file label >>
TOTALBLKS:=DATABLKS+LABBLKS;  << blocks used by both >>
```

Extent Considerations

At this point, we can determine the extent size (in blocks or in sectors) of each file extent. The formula is (given MAXEXTS is the maximum number of extents requested by the file creator at creation time) as follows:

```
EXTSIZE'BLOCKS:=TOTALBLKS/MAXEXTS;    << in blocks >>, or
EXTSIZE'SECTORS:=EXTSIZE'BLOCKS*BLKSIZE;    << in sectors >>
```

For our purposes, we will use the EXTSIZE'BLOCKS-in-blocks formula. Now, let us digress for a moment. As we have said before, the maximum number of extents of a given file can be specified on a :BUILD or :FILE command, and defaults to 8. But, the maximum number of extents actually granted (this is NOT the number of extents actually used!) may be smaller than the maximum number of extents requested in this way! In order to explain the reason for this, we must first recall a fact that will be of

paramount importance to us in this entire discussion: ALL EXTENTS OF A FILE MUST BE OF THE SAME LENGTH, EXCEPT THE LAST ONE, WHICH MAY BE OF SMALLER LENGTH. Let us suppose that you try to :BUILD a file with 100 blocks and 16 as the maximum number of extents (for instance, with an MPE command like :BUILD MYFILE;DISC=100,16). Now the file system must fit an integer number of blocks into one extent. Now, how many blocks can fit into one extent? Well, the number is 7 (the ceiling of 10 0/16). But, by the rule stated above, all extents of a file except the last one must be of the same length. Thus, each extent except the last one must be 7 blocks long. But, only 14 such extents can fit into 100 blocks, leaving 1 2-block extent! So, the file system can not possibly grant you a maximum number of extents larger than 15, even though you asked for 16! n short, the "real" number of maximum extents granted turns out to be:

```
REALMAXEXTS:=TOTALBLKS/EXTSIZE'BLOCKS;
where TOTALBLKS and EXTSIZE'BLOCKS were defined above.
```

where TOTALBLKS and EXTSIZE'BLOCKS were defined above.

Now, the above statements have yet to use the END OF FILE parameter. Nevertheless, this parameter is a vital one to our calculation. It permits us to determine another crucial factor, the number of extents currently used (USEDEXTS), through the following formula:

```
USEDEXTS:=(LABBLKS+EOF/BLKFACT)/EXTSIZE'BLOCKS;
```

The above takes the number of blocks actually used by the file and divides it by the number of blocks per extent, thus getting the number of extents actually used. Now, we have the answer: if the number of extents used is the real maximum number of extents, i.e., all of the file's extents are allocated, the number of sectors used can be found by the following:

```
SECTORS:=TOTALBLKS*BLKSIZE;
```

If, however, some of the extents of the file remain unallocated, we can find the number of sectors used with this formula:

```
SECTORS:=IF USEDEXTS=REALMAXEXTS THEN
           TOTALBLKS*BLKSIZE  << if all extents are allocated >>
         ELSE
           USEDEXTS*EXTSIZE'BLOCKS*BLKSIZE;
```

The Facts in a Nutshell

In short, the above rantings and ravings boil down to the following algorithm:

```
(variables:
MAXEXTS   = maximum number of extents requested
RECSIZE   = record size (in words) of the file
BLKFACT   = blocking factor of the file
FLIMIT    = the file's file limit
EOF       = the file's end of file
ULAB      = the number of user labels allocated in that file)
BLKSIZE:=(RECSIZE*BLKFACT)/128;
DATABLKS:=FLIMIT/BLKFACT;
LABBLKS:=(ULAB+1)/BLKSIZE;
TOTALBLKS:=DATABLKS+LABBLKS;
EXTSIZE'BLOCKS:=TOTALBLKS/MAXEXTS;
REALMAXEXTS:=TOTALBLKS/EXTSIZE'BLOCKS;
USEDEXTS:=(LABBLKS+EOF/BLKFACT)/EXTSIZE'BLOCKS;
SECTORS:=IF USEDEXTS=REALMAXEXTS THEN
           TOTALBLKS*BLKSIZE
         ELSE
           USEDEXTS*EXTSIZE'BLOCKS*BLKSIZE;
```

Let us analyze an example case (you can verify it yourself!):

```
MAXEXTS  = 8 extents
RECSIZE  = 40 words
BLKFACT  = 3 records per block
FLIMIT   = file limit of 10000
EOF      = 4600 records
ULAB     = 0 user labels
BLKSIZE              := (40*3)/128 = 1 sector;
DATABLKS             := 10000/3 = 3334 blocks;
LABBLKS              := 1/1 = 1 block;
TOTALBLKS            := 3334+1 = 3335 blocks;
EXTSIZE'BLOCKS       := 3335/8 = 417 blocks;
REALMAXEXTS          := 3335/417 = 8 extents;
USEDEXTS             := (1+4600/3)/417 = 4 extents;
SECTORS              := since USEDEXTS (4) <> REALMAXEXTS (8), then
                        4*417*1 = 1668 sectors;
```

APPENDIX C

A Summary of Methods to Save Disc Space

The following is a summary of some of the possible methods of saving disc space without deleting files or making them unreadable (methods are arranged in order of descending average percentage savings):

% savings	method
25%-75%	Convert source files to QEDIT/ROBELLE format; this format is very efficient in terms of disc space usage yet still readable by compilers.
25%-50%	Convert data/COBOL files to variable record length; this can be accomplished with EDITOR's /SET VARIABLE command.
0%-50%	Improve blocking factor of files; a file's block size should be a multiple of 128 words or disc space will be wasted.
0%-25%	Set file limit of files to end of file; if the file limit of a file is not its end of file disc space is probably being lost. Note that for data files, the file limit should be greater than end of file to allow for expansion.

These operations can be performed on files one by one, or en masse using MPEX/3000.

APPENDIX D

A Summary of Methods to Speed Up File Access

The following is a summary of some possible methods of speeding up disc file access, arranged in order of descending average percentage savings of file access time:

% savings	method
50%-95%	Use MR NOBUF access for file reading/writing; for easy use of this access method, David Brown's FAST I/O routines are suggested.
5%-10%	Increase the block factor of files; this will increase the block size, and thus the buffer size of files accessed with buffering, and thus decrease the number of disc I/Os needed to access them. Make the block size as large as possible, but no more than 8,000. Set BUF=1 (only 1 buffer) to avoid getting file system error 57 (OUT OF VIRTUAL MEMORY).

APPENDIX E

Related Papers / Useful Programs

As we could not (and never intended to) say everything there is to say about disc files, we would like to refer you to the following useful reference documents and utility programs:

PAPERS

"Another MPE Feature (BUG)." A discussion of a bug in Multi-Record file access by Vladimir Volokh, VESOFT Consultants. *SCRUGletter*, Volume 4, Issue 1 for January 1981.

"How to Avoid Problems With MPE Carriage Control (CCTL)." All there is to know (well, almost) about Carriage Control. Robert M. Green, Robelle Consulting Ltd., 27597-32B Avenue, Aldergrove, B.C. Canada V0X 1A0.

HP3000 Computer Systems MPE Commands Reference Manual. Section VI — MANAGING FILES.

HP3000 Computer Systems MPE Intrinsics Reference Manual. Section III — ACCESSING AND ALTERING FILES.

HP3000 Computer Systems MPE IV Intrinsics Reference Manual. Section III — INTERPROCESS COMMUNICATION AND CIRCULAR FILES. Section X — ACCESSING AND ALTERING FILES.

"Privileged Mode — Use and Abuse." What is privileged mode and how to use it safely by Eugene Volokh, VESOFT Consultants. *SCRUGletter,* Volume 4, Issue 4 for June 1981.

SOFTWARE

"FAST I/O (aka BLOCKED I/O)." A product that permits fast, easy MR NOBUF file access available from EASY Software Co., 410 Chipeta Way, Research Park, Salt Lake City, UT 84108.

"MPEX/3000." Many useful extensions to MPE available from VESOFT Consultants.

"QEDIT/ROBELLE." A superior editor, with disc space-saving features available from Robelle Consulting Ltd., 27597-32B Avenue, Aldergrove, B.C. Canada V0X 1A0.

APPENDIX F

Cryptic File System Error Message De-Crypted

In addition to its other failings, the System Intrinsics Manual does not explain the exact reason for and/or work-around for most file system errors. In fact, most file system error messages are very hard to understand. The following is an attempt at an adequate explanation of the causes, effects, and work-arounds for different file system errors that pertain to disc files:

0 END OF FILE (FSERR 0): This error is encountered when a program attempts to read beyond the end of file or write beyond the file limit. WORKAROUND: Change the program or the file.

1 ILLEGAL DB REGISTER SETTING (FSERR 1): Should never occur for non-privileged mode

programs. For privileged mode programs, this means that the programmer attempted to do an FFILEINFO, FGETINFO, FOPEN, or FRENAME in split-stack mode (i.e., after a call to the EXCHANGEDB or SWITCHDB procedures). WORKAROUND: Do not perform the function in split-stack mode.

2 ILLEGAL CAPABILITY (FSERR 2): A function that requires privileged mode capability (e.g., open file for NOWAIT I/O, open file for EXECUTE access, etc.) was attempted without privileged mode capability. WORKAROUND: Enter privileged mode before executing the function or do not attempt to execute it at all.

8 ILLEGAL PARAMETER VALUE (FSERR 8): Parameters specified on the FOPEN call are mutually contradictory; for instance, an attempt to open a file NOWAIT on a serial disc was detected, or the program tried to open a new KSAM file without specifying the FORMALDESIGNATOR or KSAMPARAM parameters on the FOPEN. WORKAROUND: Correct the parameter.

9 INVALID FILE TYPE SPECIFIED IN FOPTIONS (FSERR 9): The file type field of the FOPEN file options is not one of 0 (STD = standard file), 1 (KSAM file), 2 (RIO file), 4 (CIR = cir cular file), or 6 (MSG = message file). WORKAROUND: Correct the file type field.

10 INVALID RECORD SIZE SPECIFICATION (FSERR 10): The record size requested was more than 32767 bytes. WORKAROUND: Specify a smaller record size.

11 INVALID RESULTANT BLOCK SIZE (FSERR 11): If the user request were honored, the block size (BLOCK FACTOR * RECORD SIZE) of the resultant file would be greater than 32767 bytes. WORKAROUND: Specify a smaller record size or block factor.

12 RECORD NUMBER OUT OF RANGE (FSERR 12): The user passed a negative record number to the FPOINT, FREADDIR, or FWRITEDIR intrinsic — this is illegal. WORKAROUND: Correct your program.

22 SOFTWARE TIME-OUT (FSERR 22): The user tried to read an empty message file or write to a full message file, an action which would cause the user to be impeded until the file stopped being empty or full, respectively (see MPE IV INTRINSICS MANUAL). However, a time out was set with the FCONTROL intrinsic (mode 4) and the request timed out before it could be honored. WORKAROUND: Do not set the time out or ensure that the request can be serviced before it times out.

26 TRANSMISSION ERROR (FSERR 26): Hardware failure. WORKAROUND: Call your CE.

30 UNIT FAILURE (FSERR 30): Hardware failure. WORKAROUND: Call your CE.

40 OPERATION INCONSISTENT WITH ACCESS TYPE (FSERR 40): The access type specified at FOPEN time does not permit this operation; fori nstance, an FWRITE is not permitted when a file is opened with ACC=IN. WORKAROUND: Specify an access type at FOPEN time that permits this operation or do not perform the operation at all.

41 OPERATION INCONSISTENT WITH RECORD TYPE (FSERR 41): It seems that this error should never show up and is merely a left-over from a previous version of MPE.

42 OPERATION INCONSISTENT WITH DEVICE TYPE (FSERR 42): The program tried to execute an operation that is incompatible with the device that it is trying to perform it on; for instance, it is trying to read the line printer or change the baud rate of a disc drive. WORKAROUND: Do not execute the operation.

43 WRITE EXCEEDS RECORD SIZE (FSERR 43): An attempt was made to write a record that would not fit in the destination file, e.g., to write a 100-byte record into a file of record length of 80 bytes. WORKAROUND: Change the file's record size, change the length of the record to be written, or open the file with the Multi-Record (MR) access option.

44 UPDATE AT RECORD ZERO (FSERR 44): The FUPDATE intrinsic (which is equivalent to the COBOL REWRITE statement) was called with the record pointer at record 0, which indicates that no record has been read and thus no record can be updated. WORKAROUND: Call FPOINT or FREAD before the FUPDATE call.

45 PRIVILEGED FILE VIOLATION (FSERR 45): A program attempted to open a privileged file (one with a negative file code; e.g., an IMAGE file) while specifying a filecode not equal to the file's filecode or while not in privileged mode. WORKAROUND: Enter privileged mode before the call or specify the correct filecode.

46 OUT OF DISC SPACE (FSERR 46): The device class on which this file resides (if this error is gotten at extent allocation time) or is requested to reside (if this error is gotten at file creation time) does not have enough contiguous disc space to accommodate this file; i.e., if NUMEXTS is the number of extents to be allocated and EXTSIZE is the size (in sectors) of one extent, this device class does not have NUMEXTS contiguous chunks of EXTSIZE sectors each. WORKAROUND: Move the file to another, less full, device class, decrease the requested file size, or decrease the extent size by increasing the number of extents in the file.

47 I/O ERROR ON FILE LABEL (FSER R 47): The internal file label of this file can not be accessed. Most likely, the file is totally cloberred and will return INVALID FILE LABEL (FSERR 108) when it is subsequently accessed. WORKAROUND: None.

48 OPERATION INVALID DUE TO MULTIPLE FILE ACCESS (FSERR 48): One of the following conditions is true: 1) The program is trying to purge (i.e., close with disposition DEL) a file that is currently loaded or being stored/restored, 2) The program is trying to rename (with the FRENAME intrinsic) a file that it does not have exclusive access to, or 3) The program is trying to open with LOCK access a file that someone else has opened with NOLOCK access or vice versa. WORKAROUND: 1) Don't purge the file or wait for the file to become purgeable again, 2) Don't rename the file or open the file with EXC access, or 3) Open the file with LOCK or NOLOCK access (whichever is the one with which the other program has the file open).

49 UNIMPLEMENTED FUNCTION (FSERR 49): The program specified an invalid parameter value in a file system intrinsic call; e.g., a disposition of 5, 6, to 7 at FCLOSE time or a file type of RIO on pre-Athena systems (ones which do not support RIO files). WORKAROUND: Correct your program.

50 NONEXISTENT ACCOUNT (FSERR 50): An attempt was made to open a file in an account which was not configured in the system. WORKAROUND: Correct the filename or build the account.

51 NONEXISTENT GROUP (FSERR 51): An attempt was made to open a file in a group which was not configured in the system. WORKAROUND: Correct the filename or build the group.

52 NONEXISTENT PERMANENT FILE (FSERR 52): An attempt was made to open a file which does not exist. WORKAROUND: Correct the program or build the file.

53 NONEXISTENT TEMPORARY FILE (FSERR 53): The program tried to open a temporary file which does not exist. WORKAROUND: Correct the program or build the file.

54 INVALID FILE REFERENCE (FSERR 54): The program tried to open a file whos e filename was invalid; for instance, the file, group, or account name was longer than 8 characters long, an invalid system-defined file was specified (e.g., $XYZZY), or no file equation was found for a back-refenced file (e.g., *MANSION with no file equation for file MANSION). WORKAROUND: Correct the filename specified. NEED

56 INVALID DEVICE SPECIFICATION (FSERR 56): The device number or device class on which the file was to be opened is not configured on the system. WORKAROUND: Correct the program.

57 OUT OF VIRTUAL MEMORY (FSERR 57): The buffer size (NUMBER OF BUFFERS * RECORD SIZE * BLOCKING FACTOR) of the file to be opened exceeds 8,192 words (or 14,000 words starting with the D MIT version of MPE). WORKAROUND: Decrease number of buffers (by specifying BUF=1 on a :FILE equation, for instance), decrease the record size of the file, or decrease the blocking factor of the file.

58 NO PASSED FILE (FSE RR 58): The program attempted to open $OLDPASS, but no $OLDPASS file exists. WORKAROUND: Correct the program or build a $OLDPASS file.

60 GLOBAL RIN UNAVAILABLE (FSERR 60): The program requested dynamic locking at file open time, but the RIN (Resource Identification Number) necessary for dynamic locking could not be gotten. WORKAROUND: Free some global RINs (with the :FREERIN command), file RINs (by closing files opened with LOCK access), open the file with NOLOCK access, or enlarge the RIN table.

61 OUT OF GROUP DISC SPACE (FSERR 61): The program tried to allocate more disc space than is allowed for a given group; e.g., it tried to build a 10,000-sector file in a group which already had 95,000 sectors and was limited to 100,000 sectors. WORKAROUND: Decrease the amount of disc space used by files in that group (by purging or squeezing files) or a ask the account manager to increase the group disc space limit.

62 OUT OF ACCOUNT DISC SPACE (FSERR 62): The program tried to allocate more disc space than is permitted for the account in which it tried to allocate it. WORKAROUND: Decrease the amount of disc space used by files in that account (by purging or squeezing files) or ask the system manager to increase the account disc space limit.

64 USER LACKS MULTI-RIN CAPABILITY (FSERR 64): The program was not :PREPed with MR (Multi-Rin) capability, yet tried to lock a file when another file (or RIN) was already locked by that program. WORKAROUND: :PREP the program with MR capability or do not try to lock a file when you have already locked another one.

71 TOO MANY FILES OPEN (FSERR 71): The program attempted to open a file, but there was not enough room in the system area (PCBX) of the program's stack for the information for that file. WORKAROUND: Close some no longer necessary files before trying the open, or run the program with the ;NOCB keyword on the :RUN.

72 INVALID FILE NUMBER (FSERR 72): An attempt was made to access (e.g., read or write) a

file that has not been opened or that is a privileged file; for instance, a read was requested against file number 10, but no file is opened as file number 10. WORKAROUND: Correct your program or enter privileged mode before trying to access the file (if the file is privileged).

73 BOUNDS VIOLATION (FSERR 73): You are attempting to read or write more data than could fit into your I/O buffer (e.g., you are trying to read 100 words into an 80-word array). WORKAROUND: Decrease the length of the data you are trying to read or write or enlarge your program's I/O buffer.

74 NO ROOM LEFT IN STACK SEGMENT FOR ANOTHER FILE ENTRY (FSERR 74): See file system error number 71 above.

90 EXCLUSIVE VIOLATION: FILE BEING ACCESSED

(FSERR 90): Exclusive access was requested to a file which is already being accessed; thus, exclusive access cannot be granted. WORKAROUND: Specify SHR (shared) or EAR (exclusive — allow read) access when opening the file or wait for the accessor to close the file.

91 EXCLUSIVE VIOLATION: FILE BEING ACCESSED EXCLUSIVELY (FSERR 91): Access was requested to a file which is being accessed exclusively by some other user. WORKAROUND: Wait for the accessor to close the file.

92 LOCKWORD VIOLATION (FSERR 92): An invalid lockword was specified at file open time or when the file system prompted the user for a lockword. WORKAROUND: Specify a correct lockword or remove or change the lockword on the disc file.

93 SECURITY VIOLATION (FSERR 93): Permitting the user to access this file in the specified access mode would be a breach of file security. WORKAROUND: Change the access mode specified in the program to one which is permitted or ask the file's creator to :RELEASE or :ALTSEC the file.

94 USER IS NOT CREATOR (FSERR 94): An attempt was made to :RENAME or FRENAME a file by someone other than the file's creator. WORKAROUND: Do not perform the :RENAME or FRENAME, ask the creator of the file to do the :RENAME, or (if you have read and write access to the file and are a user of MPEX/3000) use MPEX's %RENAME command.

96 DISC I/O ERROR (FSERR 96): Hardware failure. WORKAROUND: None.

100 DUPLICATE PERMANENT FILE NAME (FSERR 100): The program tried to save (close with SAVE disposition) a new or temporary file as a permanent file, but a permanent file with that name already exists. WORKAROUND: Purge the other file with that name.

101 DUPLICATE TEMPORARY FILE NAME (FSERR 101): The program tried to save as temporary file (close with TEMP disposition) a new file, but a temporary file with that name already exists. WORKAROUND: Purge the other temporary file with that name.

102 DIRECTORY I/O ERROR (FSERR 102): The directory (or part of it is cloberred. You're in big trouble. WORKAROUND: None.

103 PERMANENT FILE DIRECTORY OVERFLOW (FSERR 103): There is no more room in the system file directory for this file (the system file directory typically allows approximately 1200 files per group). WORKAROUND: Purge some of the files in the group in which you wish to build the file.

104 TEMPORARY FILE DIRECTORY OVERFLOW (FSERR 104): There is no more room in your job/session temporary file directory for this file. WORKAROUND: Purge some temporary files or :RESET some :FILE equations or :CRESET some :CLINE equations.

105 BAD VARIABLE BLOCK STRUCTURE (FSERR 105): The variable record length file being accessed has an inconsistent structure or would have an inconsistent structure if this access were to go through (if you are writing NOBUF). WORKAROUND: If you are writing NOBUF, correct your program; otherwise, none.

106 EXTENT SIZE EXCEEDS MAXIMUM (FSERR 106): The program attempted to build a file which would have extents larger than 65534 sectors, the maximum permitted. WORKAROUND: Increase the number of extents in the file or decrease the extent size by decreasing the record size or file limit of the file.

107 INSUFFICIENT SPACE FOR USER LABELS (FSERR 107): The maximum number of user labels for a file is 254. WORKAROUND: Decrease the number of user labels requested by the program.

108 INVALID FILE LABEL (FSERR 108): The file is inaccessible because the file is invalid (probably irrecoverably destroyed). WORKAROUND: None.

109 INVALID CARRIAGE CONTROL (FSERR 109): The program tried to do a write with a CCTL code of 1 (imbedded CCTL) but with a buffer length of 0; or, the program attempted an FCONTROL mode 1 (transfer CCTL code) with a parameter of 1. WORKAROUND: Correct the program.

110 ATTEMPT TO SAVE PERMANENT FILE AS TEMPORARY (FSERR 110): An attempt was made to close a permanent file with temporary

(TEMP) disposition; this is illegal. WORKAROUND: Correct the program.

148 INACTIVE RIO RECORD (FSERR 148): An FPOINT, FREADDIR, or FSPACE positioned the record pointer at an inactive record in an RIO (Relative I/O) file. WORKAROUND: None necessary.

149 MISSING ITEM NUMBER OR RETURN-VARIABLE (FSERR 148): An item number was specified without a corresponding variable or vice versa in an FFILEINFO intrinsic call. WORKAROUND: Correct the program.

150 INVALID ITEM NUMBER (FSERR 15 0): An item number specified in an FFILEINFO intrinsic call is invalid. WORKAROUND: Correct the program.

151 CURRENT RECORD WAS THE LAST RECORD WRITTEN BEFORE THE SYSTEM CRASHED (FSERR 151): The current record in the MSG (message) file was the last one written before the system crashed and may contain invalid information.

ACKNOWLEDGEMENT

Praises and kudos GOTO: Robert Saunders (of the HP lab) for much important information; Vladimir Volokh (of VESOFT Consultants), Robert Green (of ROBELLE Consulting), and many others for comments, questions, criticisms, suggestions, and overall moral support.

Data Communications Troubleshooting

Pete Fratus
Information Networks Division
Hewlett-Packard Company

PREFACE

Data communication problems can be extremely difficult to solve. They can also be solved very simply. Why the differences? Let's look at modern medicine for a few examples.

A patient complains of a sore arm. The doctor takes his temperature (they always take your temperature), examines his arm, asks some questions about past medical history and sends him to X-ray. Looking at the x-rays, she sees an obvious crack in the bone and places a cast on his arm. That was a fairly simple solution. Now take the same patient back to 18th century Europe. Tools for diagnosing broken arms were lacking, but there was always blood-letting. If that didn't work, the doctor could try irritants, Phrenology, magnetism and magic.

Had the doctor possessed the proper tools to do the job, the time between complaint and correct treatment would have been shortened considerably.

The type of problem, the tools available, and the technique applied can shorten or lengthen the time required to solve the problem. This presentation will help you understand the problem, become aware of the tools, and improve your techniques.

From the viewpoint of most computer users, there are four types of malfunctions. They are usage, protocol, digital and analog. Usage problems are those arising from improper use of an otherwise working data comm link. Protocol problems go beyond the users' immediate control and involve the software that handles the link. Digital problems involve the interface between the data terminal equipment (DTE) and the data communications equipment (DCE). Analog problems are limited to the data communications facility, which is the wires between the modems or data sets.

There are many approaches to troubleshooting. The process of elimination by replacing equipment, stepping through software, and circuit checks by the halving algorithm are some ways. Symptomatic troubleshooting does not eliminate any of these methods, but it does reduce the time necessary by quickly pointing out the area of the malfunction.

HEATERS AND AIR CONDITIONERS 24-23

SYSTEM COOLS INTERMITTENTLY

Symptoms	Possible Causes
Electrical	
1. Unit operates intermittently.	1. Defective fuse, relay blower switch, or blower motor.
2. Clutch disengages prematurely during operation.	2. Improper ground, loose connection, or partial open in clutch coil.
Mechanical	
1. System operates until head pressure on builds up at which time clutch starts slipping; may or may not be noisy.	1. Compressor clutch slip.

Example: Troubleshooting guide from auto repair manual.

An example of symptomatic troubleshooting can be found in nearly any automobile repair manual. You may find a flowchart or table in which the axes are labeled SYMPTOM and PROBABLE CAUSE. The idea is to find the probable causes for the condition (or problem) encountered, then by testing or a process of elimination, discover the remedy. Newer methods have been developed which can suggest solutions.

AUTOMATIC TRANSMISSION 21-30

General Diagnosis Chart

Symptoms	Causes (see list below)								
	1	2	3	4	5	6	7	8	9
HARSH ENGAGEMENT FROM NEUTRAL TO D OR R	X			X					
DELAYED ENGAGEMENT FROM NEUTRAL TO D OR R		X		X	X	X	X	X	X
RUNAWAY UPSHIFT		X		X		X		X	
NO UPSHIFT		X		X		X	X		
3-2 KICKDOWN RUNAWAY		X		X		X			
NO KICKDOWN OR NORMAL DOWNSHIFT				X					
SHIFTS ERRATIC		X		X		X	X	X	X
SLIPS IN FORWARD DRIVE POSITIONS		X		X		X	X	X	X

1 Engine idle speed too high
2 Hydraulic pressures too low
3 Low-reverse band out of adjustment
4 Valve body malfunction or leakage
5 Low-reverse servo, band or linkage malfunction
6 Low fluid level
7 Incorrect gearshift control linkage adjustment
8 Oil filter clogged
9 Faulty oil pump

Example: Symptom/Cause Chart from auto repair manual

Let's get back to the computer. Think of all of the possible problems one can have with a data communications network: hangs, disconnects, errors in the data, delays, retries, and on and on. What could be the possible causes of these problems? Noise, fade, delay, program bugs, faulty equipment, operator error, and more can all cause aggravating malfunctions.

This is what you need to know to solve these problems in a timely manner:

A. The Basics — what is the environment, what was supposed to happen
B. The Symptoms — what did happen
C. The Causes — why
D. The Tests — what tests will give the right information
E. The Tools — what tools will do the tests
F. The Solution — what action to take

This presentation will help you learn how to get from A to F.

EXCERPT

Let me give you an example to show how knowledge of the symptoms, tests, and tools can make problem solving easier. A problem occurred at a site where the symptoms were terminal hangs and garbage on the screen sometime after the session started. There was never any problem signing on. This was a point-to-point terminal on a switched line using BELL 212A modems.

Three things were done in an attempt to resolve the

problem: the MPE I/O configuration was checked, the modem options were verified, and a 1640 data scope was put into the line. The 1640 showed that a DC1 was received followed by garbage. This was either printed as garbage or hung the terminal. The assumption that followed was that something was wrong with the terminal. The configuration of the terminal showed that it was "providing clock," as was the 212 modem. At this point, it was decided that the modem options listed in the Data Comm Handbook must be wrong.

This bit of troubleshooting had gone way off on a wild goose chase. No attempt was ever made to test the most basic part of the network, the telephone line. Simple modem self-tests and loop backs were completely ignored. The 1640 served no useful purpose at this point.

A more reliable approach would have been to start by defining the exact symptoms, determining the possible causes, and making some appropriate tests. Using the new information gained through this technique, troubleshooting would have been more directly related to the problem.

The Basics

HP262X terminal
Point-to-point terminal connection to a port
Switched public line
Full duplex modem with good complement of diagnostics

The Symptoms

Apparently random terminal hangs and garbage occurs only on one line

The Causes

Fortuitous line problems
Faulty modem
Faulty terminal

The Tests

Modem loop back with test pattern
Modem self-test
Terminal data comm test

The Tools

None needed

The Solution

Switched lines are susceptible to noise and other problems. Since each new connection uses a different route, conditioning is not available (and would not help noise anyway). Therefore, the terminal should be reset if it is hung or the data retransmitted if it was garbaged. If that doesn't help, redial the connection.

SUMMARY

This excerpt is an example of what my presentation will cover in much more detail. I am currently working on flowcharts and decision tables to make solving data comm problems easier by encouraging the use of symptomatic trouble shooting. This should lead to using the proper tools in the proper order.

Financing Quality Solutions

Melissa J. Collins

Is your manager a fire-breathing dragon? Does your budget get thrown in the dungeon year after year? If you answered "yes" or even thought twice about one of these questions, you are not alone.

The Data Processing manager faces many challenges and pitfalls in operating his or her department. One such pitfall is the budgeting and finance area. This paper will discuss solutions and methods to deal with department monies (or lack thereof) and interaction with a non-technical manager.

THE SUCCESSFUL DATA PROCESSING DEPARTMENT EQUATION

Good Equipment + Good People + Money = Quality Solutions

All of you have made an excellent choice in equipment. If you don't have the good people, they are out there for the hiring. Now all you need is the money and the management's support and Quality Solutions will be within your grasp.

So where does the DP manager fit into the money part of the equation? The manager submits a budget of his monetary wants/needs for a fixed period of time. Of course, just because he asked for a million dollars, doesn't mean he gets that amount. The DP manager must convince the upper management that the monies requested are sound investments in the company's goals and futures. This is where the hard part comes into play. HOW DO YOU convince the upper management, who has little or no computer training, that your budget goals are not unrealistic?

GETTING YOUR BUDGET APPROVED

First of all, you must face three facts. Once you come to grips with them, the outlook will not be so gloomy.

1. Your management is not against you or your department.
2. Getting the monies necessary to finance any DP project is sometimes harder than bleeding a rock.
3. Anything truly worthwhile is worth a small battle.

The first step to insure approval of your budget is to be realistic about your requests. But, at the same time, do not under-budget your department. This is a fine line to walk, but it can be done. It is always nicer to come in under budget than over budget, but if a department is consistently under budget, then a manager can be accused of "padding the budget." A few guidelines to consider about budgeting items other than normal expenditures (salaries, maintenance contracts, consumables, etc.):

1. If there is the slightest chance that you will need a new piece of equipment, budget for it.
2. Be sure you have sufficient justification for new equipment.
3. If using the budget as a tool for justification of new employees, provide good evidence of need. (Such as project time tables, department workload, etc.)
4. As a tradeoff — instead of new employees, budget for programmer productivity tools whenever possible. Offer this as an alternative to your manager. If the same results are achieved, the lower cost alternative will always be chosen.

If a manager is dealing with a non-DP superior, having his support is very helpful. The time you spend educating a non-DP manager is time well spent. If your manager knows what a disc is and what its uses are, getting approval for a new one is not quite as painful.

The Data Processing department is surrounded by an aura that threatens some managers. The high technology and computer "buzz words" are enough to scare off anyone who doesn't know what is going on. By working with your manager and educating him, you will find that he will support you more. The old adage, "You can lead a horse to water, but you can't make him drink," applies here. Some managers could care less about learning more about the DP department. Subtle tactics can be used to educate a non-technical manager. Such tactics include, but are not limited to, inviting your manager to participate in your weekly staff meetings, taking him on a tour of your facility BEFORE you present your budget requests, or taking him to a regional user group meeting. These actions may prick his interest to learn more about the DP department.

Interaction with users may not seem really important in attaining your financial goals. But consider this; if the users are unhappy with the DP department, this attitude will filter up to the managers of said users. The managers will, in turn, convey this attitude to the higher management who may ultimately be in the position to approve or disapprove your budget. You can't expect your users to be happy all of the time, but shoot for making them happy as much as you can. It will help around budget time.

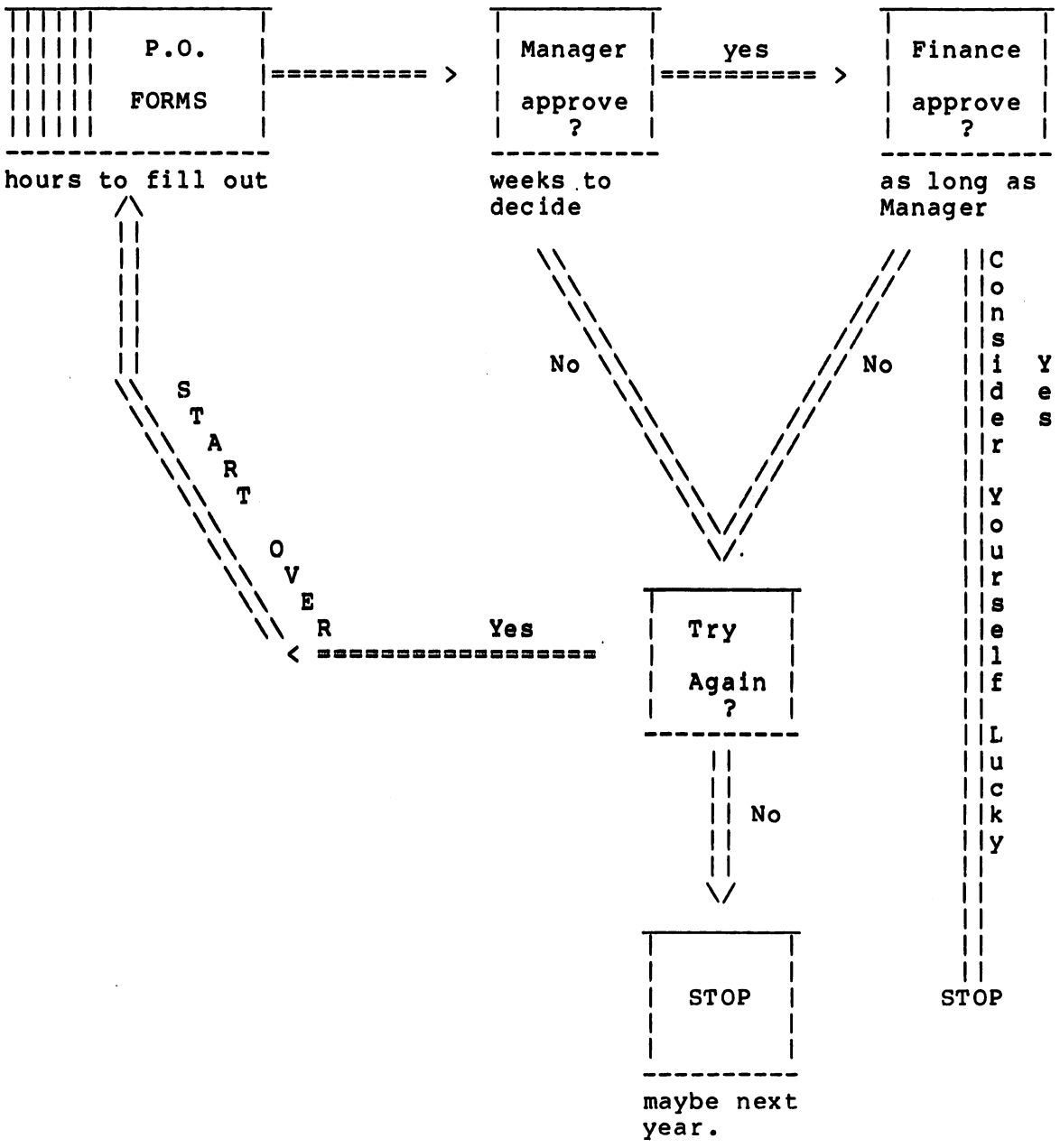
USING YOUR BUDGET FUNDS

Now that your budget is approved, you don't have a

free reign in distributing the monies awarded to your department. This is a fact everyone has to face. This problem has a name well known to all of us — RED TAPE. The government doesn't hold a monopoly on the man-made phenomenon. But, there are ways to circumvent its nasty powers. Take purchase orders for instance.

Instead of filling out two tons of paperwork for a purchase order number, have the vendor submit a bill to you that you can sign off. The end result is the same; you get product/services and the vendor gets the money (probably in shorter time). If you can follow the chart below, you can trace the path of the purchase order request. If a bill was sent to you, you could eliminate all of these steps.

PURCHASE ORDER PROCESS



Most, if not all, managers have a monetary limit for which they are allowed to sign without having their managers approval. Using this limit will help you eliminate not only the purchase order request cycle, but will also help you acquire products/services without the approval cycle. If you want a vendor's product that is $7,500, but don't have said product in the budget for this year, don't give up hope. Use the art of finagle. First

and foremost, check your budget very carefully to see if you have any extra money anywhere. If there is some money, but not enough, consider cutting something out, like a tape cabinet. Then, working with the vendor, try to agree on some financial arrangement that will get you your product, the vendor his money, and, most important, not get you fired. Most vendors will work with you on this. Instead of not making a sale, they will gladly bill you on an installment plan. As long as these installments are under the amount you are allowed to sign for, you will all be getting what you want.

However, it is not wise management to practice the art of finagle all of the time. The more you use it, the more likely you are to get in trouble. It is to be used when the political climate of your company is not conductive to spending, or when you know that there is no other way to get what you want. Dealing "under the table," so to speak, is a tool you have available to you as long as you do not abuse it.

FIGHTING BATTLES

So what if it's in your budget? Your manager could change his mind by the time the purchase order hits his desk. What recourse do you have except retreat? You can stand up and fight for what you want, diplomatically of course.

> Di-plo-ma-cy (di plo'me se) n. 1. the conducting of relations between nations 2. the skill of doing this 3. skill in dealing with people; tact SYN. see TACT

According to Webster, the art of diplomacy is based on the skill of dealing with people. For a manager to achieve his goals, he must know how to deal with people. When a proposal is rejected, it can be very difficult for a manager to understand why his request was turned down. The first reaction is usually one of anger or frustration, which if vented on your manager, will leave little or no chance for a reverse decision.

When you have an important request turned down, analyze the situation. What was the reasoning behind the decision? The political climate of your company may have wreaked havoc on your proposal. Or you did not justify it in a way that your manager could appreciate your true need for your request. Try to see the situation from your manager's point of view. Discuss it with him, and ask him why the request was turned down. Try not to put him on the defensive about his actions. If any of his doubts can be resolved, do so as soon as possible while the situation is fresh in his mind. If this fails, you still have ways in your grasp to continue the fight.

The battleground is already set. You want something you feel you need, and your manager told you that you cannot have it. Whatever his reasoning was for denying your request, if you still feel very strongly that you need this request, sound the charge.

For an example, you have requested $3750.00 for a new memory board and your manager says no. Examine your reasons for this request. Obviously you feel the machine is slow, and your users are starting to recognize this fact. With your continued development work, the situation will only grow worse. Direct a memo to your manager explaining the situation as clearly as possible. Include in your comments that the users are starting to show dissatisfaction with the response time of the machine and that the situation will get worse. Other points to consider for mention are a slump in productivity and decreased throughput. If your General Ledger is coming up on end of month close, illustrate the consequences of productivity slumps and decreased throughput. That will make him stand up and take notice.

The memo serves two purposes: one, it informs your manager of the consequences of the denied request, and, two, should the situation not resolve itself, when the complaints start pouring in, you have documented proof that you tried to rectify the situation.

When the users do start to complain, and they will, tell them what the problem is and what you have tried to do about it. Be sure that you do not, under any circumstances, criticize your manager in this discussion. Eventually, the user attitude will be translated up the line to other managers and the pressure will be shifted onto your manager. This is the long way to go around the mountain, but you will eventually get what you want/need without stepping on anyone's toes.

Should this approach fail, give your manager alternatives. Instead of getting another memory board, consider optimizing your machine. In presenting this concept to your manager, provide as many solutions as possible. You could hire another programmer to optimize all of your code. You could buy OPT/3000* and train one of your staff to interpret its data and optimize where necessary. Or you could hire a consultant and let him figure it out. All of these are alternatives. When presented to your manager, he will realize that these are all costly alternatives, much more than a memory board. So in order to save the company money, you will most probably get your original request.

If all other efforts fail, start a memo blitz. Every week or so, send your manager yet another memo concerning the subject. Be sure that these memos are inoffensively worded or the heat will turn against you. You may get your request just because your manager wants the paper barage to stop or he doesn't want to be bothered anymore.

One final word on the subject. If you are informed that the answer is still NO after all of your efforts, it may be best to hold off for a while until the smoke has settled and then try, try again.

---

*Not to be considered an endorsement of this product.

Tips and Techniques in Writing for the HP3000 IUG Journal

Dr. John R. Ray
Editor, HP3000 IUG Journal
University of Tennessee
Knoxville, Tennessee

Dr. Lloyd D. Davis
Associate Editor, HP3000 IUG Journal
University of Tennessee
Chattanooga, Tennessee

WHY WRITE?

If you are not a writer by profession, you may be hesitant about writing for a professional publication such as the HP3000 IUG Journal. The fact of the matter is that your professional skills are more important than your writing skills. Like many professional publications, The Journal builds its reputation on being written by professionals in the field for other professionals.

If you have experience, then we encourage you to share your knowledge through a Journal article. To help you get your thoughts down on paper, we have put together some tips on writing for non-writers. The professional and personal benefits derived from writing an article are of major importance.

What are the benefits to be derived from writing an article? For one thing, having an article published in the Journal generates publicity for both the author and the author's firm. The author benefits by being recognized as having expertise on the topic. Your firm benefits by being recognized as having leading professionals on its staff and by having its name brought to the attention of professionals nationwide.

Another benefit of writing an article is the personal satisfaction that comes from having contributed to the betterment of the profession through sharing your knowledge. There is also the satisfaction of seeing your name and your ideas in print.

THE REVIEW PROCESS

Reviewers evaluate all articles basically on content, not grammar or literary style. For each article, the reviewers fill out an evaluation form and recommend that the article be either: (1) published; (2) revised and published; (3) revised and reviewed again; or (4) not published. Articles that are original, timely and previously unpublished, devoid of sales or promotional material, and of national interest and value to a significant number of readers have the best chance of being published.

The review process usually takes four to six weeks. As soon as the reviewers' evaluations are received, the author is notified of their decision. If the reviewers recommend that an article be revised, the author will be provided with specific recommendations.

If the information in the article could easily become dated, the author should note this in a letter attached to the article when the article is submitted for publication. The staff will then make a special effort to publish the article before the information becomes out of date and will, if necessary, contact the author for updated information immediately before publication.

Most articles will require some degree of editing before publication. The staff may suggest refinements in the areas of literary style and organization. If there are corrections in the areas of spelling, punctuation, grammar, or word choice, these will be noted on the article. The article with annotated remarks will be returned to the author for approval prior to publication unless editorial changes are minor.

On occasion, the staff may telephone an author and ask a question about the information in an article. Although the editorial reviewers do review articles for accuracy of information, the author is still responsible for the accuracy of his or her article. Also, publication of an article does not mean that the ideas expressed in the article are endorsed by the HP3000 International Users Group and/or its Journal editors.

MANUSCRIPT REQUIREMENTS

Most articles submitted for publication in the Journal are four to eight double-spaced, typewritten pages, but articles longer and shorter than this have been published. Very long articles of twenty pages or more may be published in parts as a series.

To estimate the number of pages an article will be when published, have the article typed with 50 to 55 characters on each line. This will give you the approximate number of lines the article will be when published. The Journal uses a two column format with about 55 lines per column. Hence divide the total line count by 110 (two columns of 55 lines make one page). This quotient is your rough page count.

If space is required for exhibits such as formulas, tables, charts, diagrams, and figures, then the page count should be revised upward to reflect that space.

All articles submitted for publication in the Journal should be double-spaced, typewritten on one side of

8½×11" white paper. Ample margins of at least 1 to 1½ inches should be left on all sides. Articles typed with about 53 characters on each line would be appreciated, but this is not mandatory. Subheadings should be inserted where appropriate in the article, but again, this is not mandatory. Footnotes, tables, and figures should be on sheets of paper separate from the article. Indicate the placement of tables and figures within the article by giving each table, figure, etc., a number and using this number within the text.

Footnoting has as its goal the conveying of necessary information to enable the reader to accurately identify the location of the material to which reference is being made within the article. The most important traits of footnoting are accuracy, completeness, and style consistency. If you are not familiar with footnoting techniques, several good style guides are available to serve as references. These include:

1. Publication Manual of the American Psychological Association, second edition 1979;
   Copies may be ordered from
   Publications Sales
   American Psychological Association
   1200 Seventeenth Street, N.W.
   Washington, D.C. 20036
2. Form and Style — Theses, Reports, Term Papers
   William Giles Campbell
   Stephen Vaughn Ballou
   Houghton Mifflin Company, Boston, 1974
3. A Manual for Writers
   of Term Papers, Theses, and Dissertations
   Kate L. Turabian
   Fourth Edition
   University of Chicago Press, Chicago, 1973.

The above references give you the standards for footnoting. In practice when writing for our Journal or most other professional journals, read the journal in question for the style of footnoting used in that journal. By utilizing the examples found therein as a guide for your required footnotes (you may not need any), you can easily handle yours with a high probability of being correct and complete.

All pages containing copy, footnotes, tables and figures should be numbered sequentially, with tables and figures being the last of the pages. Numbering pages is important in case the pages do get out of sequence.

Black and white photographs to accompany the article are welcomed. An explanation of each photo (a caption) should be submitted with each photo. A caption can be written on the back of the photo or on a sheet of paper. If written on a sheet of paper, the sheet of paper should be numbered as the last page of the article, and if there is more than one photo, the caption should be numbered to indicate with which photo it is associated. Photos cannot be returned.

A short author's biography, including title, firm, membership in professional societies, special accomplishments and honors, should be submitted with the article.

If the article has been submitted to another publication or has been previously published, this should be noted in a letter accompanying the article. As mentioned before, if the information in the article can become dated soon, then this also should be noted.

Before mailing your article, read it over carefully. Recheck all figures and mathematical computations. Sometimes mistakes occur in typing. Remember, you are responsible for the accuracy of your article.

After you are certain that your article is accurate and to your liking, make a copy of it. Keep one copy for your records and mail the other to the Journal. All articles submitted to the Journal and all correspondence regarding publishing in the Journal should be addressed to:

John R. Ray
University of Tennessee
Dept. of Curriculum and Instruction
312 Claxton Education Building
Knoxville, TN 37996-3400

TIPS ON WRITING FOR NON-WRITERS

Often the task of writing seems too formidable to undertake. The ancient Chinese proverb states "Each journey of a thousand miles begins with the first step." This is also true in writing. You must eventually start placing words on paper or equivalently on other media. But how do you get started? We have listed several points that we believe will be helpful to those without previous, extensive writing experience.

1. Have something to say. When writing an article for the Journal, what you have to say is more important than how you say it. Ask yourself what subject you want to write about and what you want to say about it.
2. Make a list of the points you want to discuss in the article, then arrange these points in the tentative order you want to discuss them. This will give you an outline of your article. Use single words and phrases rather than complete sentences. If the list becomes too long or unwieldy, the subject may be too broad to be covered in the space of one article. In such a case, limit the subject and eliminate several lesser points.
3. Ask yourself who, what, when, where, why and how. This is another method of preparing an outline for your article. You could also compile a list of questions you will answer in the article.
4. Pretend you are writing a long business letter on the subject or preparing a report for your firm.
5. Write as you speak. If you have difficulty getting your thoughts down on paper, try dictating them into a dictaphone or tape recorder.

6. Don't worry about saying it right the first time. Concentrate on your thoughts, not the words. Once your thoughts are written down on paper or transcribed from a dictaphone or tape recorder, you can go back and revise your wording.
7. Try "The purpose of this article is . . ." if you have difficulty starting the article. You can change this first sentence later if you want, although this is an acceptable way to begin an article. "In summary" and "in conclusion" are acceptable and easy ways to end an article.
8. If all else fails, consider having a professional writer write your article for you. Your firm may have a public relations firm on retainer or a public relations writer on staff you could use. Do make sure, though, that you provide the writer with in-depth information that is current and topical and that you review the article for accuracy and value upon completion. Otherwise, the article probably will not be of interest and value to the readers of the professional (HP3000) journal and therefore runs the risk of not being suitable for publishing.

FORMAT

There is no absolute format, standard, or arrangement that must be followed in preparing an article for a professional journal. Items that might be appropriate for one article might be totally inappropriate in another. After the author has selected those things about which to write, the format or physical arrangement (headings) can be determined. To aid the writer, sections with appropriate headings and subheadings might be selected from the following list (in about the same order):

- Report Title
- Introduction
  - Background
  - Problem Statement
  - Information Sources
- Procedures
  - Design of experiment or solution
  - Sample Selection
  - Equipment
  - Measures Used
- Findings (Data)
  - Presentation of facts and data
  - Interpretation of findings
  - Limitations of "facts" meanings
- Summary and Conclusions
  - Short restatement of goals for article
  - Brief statement of findings
  - Any conclusions
  - Suitable recommendations
- References (Bibliography)
- Appendix (if any)

Again we stress that the above list is a very formal list of topics that might be found in some research papers. Rarely would all of these be found in the average journal article. However, some of these may provide an outline or skeleton upon which you may structure your writing and aid you in a readable, logical organization for your paper. Select from the topics on the list a topical outline that suits your article; utilize headings and structure to augment or replace these topics as your article requires.

STYLE AND READABILITY

Articles for professional journals sometimes suffer from being too stiff and rigid and/or from being awkwardly worded. Authors should strive for a style that is clear, direct, and effective. Word choice should be appropriate for the populace that reasonably might be expected to read the article. Therefore word choice should be chosen so as to both convey the problem and its solution and as well not require the reader to use a dictionary for frequent translation. Articles should be written in a direct, straightforward manner without being overly elaborate and structurally complex. Although, as earlier mentioned, there are writing conventions common to writing for professional journals, these should not interfere to the point of making good writing bad. Rather each author should utilize his/her individual skills in communications to convey meaning to the reader. Several methods (3:41-3) for improving readability follow:

1. Appeal and interest increase readability.
2. Personalization means putting human interest into the report: through a review of previous investigations as a story of other persons' successes and failures, an account of how the author collected and treated the data, illustrative cases, and deviations from central tendencies.
3. Pattern or design should be made plain to the reader.
4. Through appropriate emphasis the reader should get the important points.
5. Too great density or concentration of ideas may make reading difficult, requiring some expansion or dilution.
6. Plain words are important in making a report readable.

Remember that style is to foster clear and effective communication, not to confuse it. Carter Good reports (2:409) "As long as young scientists and scholars write accurately, clearly, and attractively, their differences in expression may render science a happier way of life for them and for the reader."

SUCCESS

We have stressed those points that we believe important in writing a journal or other professional article. Many of these are somewhat mechanical and pro forma; others are good sense types of points. It all requires an idea, a suggestion, a fresh point of view, something important enough to justify your writing and others reading. You may say, "But no one has ever heard of me

before. What chance have I to write something and see it published?" Not surprisingly, what you have to share and say is more important than who you are or where you are from. Thomas Frantz (3:384-386) surveyed the editorial boards of six professional journals and asked these editors to rank order criteria commonly used in evaluating manuscripts for journal publication. His findings follow in tabular form.

TABLE 1

Summary of 14 Criteria for Evaluation of Manuscripts Ranked in Importance by 55 Members of the Editorial Boards of Six Journals

Criteria	Mean Rank	S.D.
1. Contribution to knowledge	1.8	1.2
2. Design of study	3.5	2.1
3. Objectivity in reporting results	4.7	2.3
4. Topic selection	5.5	2.9
5. Writing style and readability	5.7	2.7
6. Practical implications	6.4	3.3
7. Statistical Analyses	6.5	2.5
8. Theoretical Model	7.0	2.7
9. Review of the literature	7.2	2.3
10. Clarity of tabular material	8.1	2.3
11. Length	10.2	1.6
12. Punctuation	11.5	1.9
13. Reputation of author	12.6	1.9
20. Instituional affiliation	13.5	0.9

The above research reports that the contribution to knowledge the article makes is of primary importance. Also, as the article reports, among the top six criteria are objectivity, topic selection, writing style and readability, and practical applications. Of least importance are the author's reputation and institutional affiliation. The moral here is that "who you are" is not important; rather "what you say, what it means, and how it reads" are all nearly equally important.

Don't get discouraged if your article is not accepted for publication. Usually, the reason an article is not accepted for publication is that it is too general in scope and does not provide enough in-depth information to be valuable to other professionals. Keep in mind that many famous authors have had articles rejected for publication but did not quit trying. As the saying goes, if at first you don't succeed, try, try again.

REFERENCES

1. Frantz, T. F. "Criteria for Publishable Manuscripts," Personnel and Guidance Journal, 47 (1968), 384-386.
2. Good, C. Essentials of Educational Research, New York: Appleton-Century Crofts, 1966.
3. Strang, R. "Principles of Readability Applied to Reporting Research," Improving Educational Research. Washington: American Educational Research Association, 1948. p. 41-43.

Management: Key to Successful Systems Implementation

Gary A. Langenwalter
Manager, MIS
Faultless Division
Bliss & Laughlin Industries
Evansville, Indiana

When I arrived at Faultless four years ago, the new on-line Order Entry system was supposed to be completely operational. I found a completed general design, some detail design, and 10 programs coded. The hardware vendor (not HP) had promised Faultless management that they would contribute one person for one year, we would do likewise, and the result would be a state-of-the-art order entry system. We finished 1½ years late, with an investment of 6+ years of effort. We are currently replacing our old hardware with an HP3000, and replacing all our software. This conversion was scheduled to take 14 months, finishing November 30, 1981. Our best current projection is August 1982. In all fairness, I must mention that the Master Scheduling package that we bought three years ago was installed on time, under budget, and it met our expectations.

We at Faultless are not alone. Consider the following three disasters, all of which occurred in Fortune 500 companies in 1980:

> "A major industrial products company discovers one and a half months before the installation date for a computer system that a $15 million effort to convert from one manufacturer to another is in trouble, and installation must be delayed a year. Eighteen months later, the changeover has still not taken place.
>
> "A large consumer products company budgets $250,000 for a new computer-based personnel information system to be ready in nine months. Two years later, $2.5 million has been spent, and an estimated $3.6 million more is needed to complete the job. The company has to stop the project.
>
> "A sizable financial institution slips $1.5 million over budget and 12 months behind on the development of programs for a new financial systems package, vital for day-to-day functioning of one of its major operating groups. Once the system is finally installed, average transaction response times are much longer than expected." (McFarlan, p. 142)

Ollie Wight, the leading consultant in the manufacturing systems field, estimates that there are fewer than 25 "Class A" MRP users in the country! That number compares poorly to the multiple thousands of companies that have tried to implement manufacturing systems, each with the intent to succeed. We are one of the "thousands"; we are working to become "Class A."

The major risks of systems implementation can be categorized as follows:

1. Failure to obtain all, or even any, of the anticipated benefits.
2. Costs of implementation that vastly exceed planned levels.
3. Time for implementation that is much greater than expected.
4. Technical performance of the resulting systems that turns out to be significantly below expectation.
5. Incompatibility of the system with the selected hardware and software. (McFarlan, p. 143)

Three factors that determine the degree of risk are listed below:

1. Project size. The larger the size, the greater the risk. Size is also relative — a $500,000 project has much greater risk for a $20,000,000 company with a 3 person MIS staff that has never installed anything of its size, than for a $200,000,000 company with 20 programmer/analysts.
2. Experience with the technology. Unfamiliarity with the computer in question, or its operating system, or database, or TP monitor and terminals increases the risk.
3. Project structure. Having clearly defined inputs and outputs, which all users agree upon beforehand substantially reduces the project risk. I have not yet seen this, but it is theoretically possible. Conversely, when people are still changing basic systems functions and designs midway through the project, that project is doomed to overrun both temporal and financial budgets. The military is particularly adept at this (aided and abetted by the contracters).

My current experience, plus my previous background as an educator and consultant with a major DP hardware vendor, support the hypothesis that forms the basis for this talk:

HYPOTHESIS

The single factor most responsible for success or failure of system implementation is management. Good management requires identification and minimization of risk of failure, plus continual execution of the three basic management principles: Planning, Organizing, and Controlling.

The implementation of a system will be successful if, and only if, it meets three basic goals which are the converse of the risks listed above:

1. On-time completion.
2. On-budget completion.
3. The completed and installed system must meet both its specifications, and the users' expectations.

Let us review in some detail how each of the basic management techniques can be used to insure successful systems implementation.

PLANNING

Perhaps the best way to approach the topic of planning is with a cursory overview of the techniques available. Both PERT charts (or CPM charts, or "Bubble charts") and bar charts have been widely used for years. Appendix B includes a sample of each. In general, computer programs are a tremendous help in handling complex PERT charts, and recalculating critical paths.

Time estimating is perhaps the biggest stumbling block to proper systems implementation time and cost projections. Various articles suggest that each person on a project be scheduled at only 70% efficiency, and that one should allow 2-3 weeks for a user decision. My own personal experience indicates that one should allow 1-2 months for vendor feedback (to an RFP, for example), and for scheduling vendor presentations and reference visits. Also, if a person is managing others, 20% of his time should be allotted for each person managed, subject to the discretion of the estimator. Finally, an estimator needs to allow "Contingency time" of 20-200%, depending on the tightness of the other estimates, and the degree of risk inherent in the project — the contingency factor should increase proportionally with the risk.

Now, down to the actual planning itself. In my opinion, the only intelligent way to implement a large system is to break it into four phases, with management, the users, and MIS mutually agreeing to the functions, cost, benefits, and time estimates at the end of each phase. This minimizes the risks involved, and maximizes the probability that the user department will implement the finished product successfully. We will examine each phase below.

1. Initiation Phase

This phase includes the preliminary survey, a rough estimate of potential costs and benefits, and the selection of the alternative perceived to be the most attractive (make vs. buy, Vendor A vs. Vendor B, etc.). It culminates in a presentation to top management of the Systems Proposal written jointly by the user department manager and the MIS Manager. If top management approves, the system implementation enters phase 2. If not, it can be reworked or dropped, with minimal expenditures of resources to date.

This phase is the most important of all; it creates the basic expectations of system functionality in the minds of users and management. It should be noted that the basic system functions are defined by the person who will use them in his daily work, not by the MIS department representative. "Systems are tools for the manager, not toys for the technician." (Wight, p. vii)

Some of the topics which are covered in the Systems Proposal (or Management Overview) are management summary, major system benefits, economic justification, and schedule. Appendix A1 contains a more comprehensive list of topics included in the Systems Proposal.

One other topic which needs consideration throughout the implementation of a new system is the fear of change of the part of some people in the company. Some will be afraid that they will lose their jobs; others that they will not be able to measure up to the new expectations; and still others that they will lose their status with their peer groups, and/or that their work groups will be reorganized. These fears, unless addressed, can result in passive or active resistance to the new system on the part of the people whose daily enthusiastic cooperation is an absolute requirement. They must, therefore, be actively addressed and overcome.

2. Analysis Phase

This phase starts with a study which examines in greater detail all major assumptions and promises of the original proposal. Greater attention must be paid to any area that includes major uncertainty (response times with the particular hardware, application, and database under consideration, for example). Cost and benefits estimates are updated with the new information. My experience indicates that costs almost invariably increase, and benefits almost equally invariably decrease. Finally, the MIS department writes the Functional Specifications for the proposed system, and has them approved by the user department(s) affected. After they all agree, they jointly present them to the Steering Committee, with updated costs and benefits. If management approves, the project continues; if not, it is either discontinued, or revised. At this stage still, there has not been a major expenditure of corporate resources.

The Functional Specifications (or General Design) document can include the major logic chart, proposed input and output layouts, a training plan, future capabilities, and a contingency plan, to name but a few

of the many topics. A more complete list appears as Appendix A2.

3. Design Phase

This phase defines how the system will be built. It is finished upon the completion of two major documents: the Design Specifications (or Detail Design), and the User's Manual.

Some topics that the Detail Design Specifications include are a detailed system flowchart, with input and output defined for each program, security, detailed program functions, specifications, and logic, and a detailed project implementation schedule. Appendix A3 contains a more exhaustive list.

One item that must be covered in appropriate detail is the Contingency Plan. All hardware, even HP's, and all software, even Faultless', will eventually fail. Such an event cannot be allowed to totally stop a critical department from functioning.

The User's Manual includes pictures and descriptions of all input and output screens, and reports, with explanations of all fields — what they mean, and how to change their contents, if appropriate. It also includes operating instructions (how to sign on to the system, what to do in case of problems, etc.). It must be written in language that the person in the functional department will easily understand.

These two major documents, plus Contingency Plan, are jointly presented by the user department manager and the MIS manager to the Steering Committee, with the re-revised cost/benefits data. If management approves, the system enters the final phase of implementation. If not, the minimum resources possible have been expended thus far; the project can be either revised or dropped. At this stage, all parties involved will have agreed on the details of the new system; there should be no "surprises" from here on out. There should be no reservations about technical capabilities, or about what the system will do.

4. Construction Phase

This phase is the one that includes the actual programming, testing, and documentation. In a well-managed project, more than 50% of the time should already have been spent designing. This minimizes changes, revisions, etc. that are the bane of efficient and effective systems. Let us discuss each subphase independently.

Programming is a complex enough topic that it warrants books, talks at this convention, and week-long training courses. Let me outline my views briefly, and then continue with the subject at hand. All programming should be top-down, structured, and modular. Each program or module must be tested and documented as soon as it is completed. It is then, and only then, that it can be included in the account that contains completed programs.

I will knowingly raise a controversy by suggesting that users should design their own screens (with V/3000, where applicable), and write their own reports (we are using REX for that purpose). To me, the data belongs to the user. Assuming that he understands the contents and implications of the numbers that exist in the database (and he should, for in most cases we hold him responsible for their accuracy), then he should be given the tools to generate the reports and inquiries that will allow him to manage his portion of corporate resources optimally. In other words, I refuse to perpetrate an "IBM" (International Brotherhood of Magicians) image with regard to my department.

The Systems Manual is a major document. It needs to follow predetermined specifications and formats, and, more importantly, must be updated throughout the life cycle of the system. There are very few things more dangerous than a slightly outdated Systems Manual in the hands of a programmer who is trying to maintain a system.

The Operations Manual is a must, whether your MIS department has a formal operations group or not. This document tells the operator how to run the batch jobs, back up the system, recover in the event of failure, where to send the output, etc. It defines expectations. If there is no formal document, the person who normally runs the job is generally the only person in the company with that information. The financial risk that represents to a company increases with the importance of the application (for example, weekly payroll).

Training cannot be overemphasized. The responsibility for training users lies with the Project Manager (the user department manager) rather than with MIS, because the head trainer becomes the person who knows the application better than anyone else in the organization. In smaller organizations, the Project Manager will train users directly; in large organizations, he will train other managers, who will then train their own people.

Training can and must commence as soon as the first few programs are finished. After the Project Team has trained itself, it is time to start familiarizing other personnel with the screens and reports they will be using soon. These people can often suggest invaluable improvements, some of which take almost no time to incorporate. The ones that involve much time must be prioritized, and approved by the Steering Committee prior to inclusion. The end users will also spot program flaws that escaped everybody else.

All user training and all program testing, except volume and response time testing, must be performed on a small test database, preferably one distilled from your real live database. My user personnel respond much more favorably to reports which include casters that they do to reports which feature bicycles.

Training is the one place that most people grossly underestimate the time and resources required for a proper implementation. Most people also underestimate

the numbers of people that must be trained, and perhaps even educated. Training materials can be acquired from the vendor, if the software is purchased, and from video-tape training companies such as ASI and Deltak.

One has three choices for final testing: Parallel testing (which works well for financial systems, for example), Pilot testing (which can be used for some manufacturing systems implementations), and None (which I cannot recommend; the only cold turkey that I like is that which is left over from Thanksgiving dinner).

Final testing also quickly unearths any latent run time or response time problems. Although painful, and embarrassing, it is better to discover those problems at this stage than to try to squeeze 25 hours of processing into a 24 hour day after the old system has been cut off!

After the final testing is complete, one faces the actual conversion. Although this sounds simple ("Just take the old data and load it into the new database."), it can be most complex. Each type of data to be converted must be examined. Each outstanding piece of paper must be considered (Do we leave it there? Replace it? How do we find them all? What about the ones we miss?). To illustrate the complexity of such a task, consider that it took us at Faultless the entire Labor Day weekend, running around the clock, to cut our MRP database over from our other (non-HP) computer to the Series III. The process involved over 30 steps. The process and programs were so complex that we ran test runs on the conversion programs themselves several times.

ORGANIZING

Since the most important person in an implementation effort is the Project Manager, let us start by briefly defining his (her) attributes and responsibilities.

The Project Manager, in my opinion, must come from the department most affected by the project (that is, the one that will gain the most if it succeeds, and lose the most if it does not). It should be the person who will manage that function on a day-to-day basis after the system is successfully installed. The MIS Manager should be Assistant Project Manager, to insure that what the user wants is technically feasible. On a major project, the Project Manager position involves a full-time effort, especially when training commences. I know that in the "Real World," those people are often totally busy just keeping the company running on a day-to-day basis. But nobody else has the intimate knowledge of how that department really functions on a daily basis that is required for successful design and implementation of the new system. Faultless top management backs this philosophy 100%, by saying that if a department is not interested enough to furnish a Project Manager, the project will not commence.

The Project Manager is responsible for writing the functional specifications at the commencement of the project. They form the basis for all subsequent development. In my opinion, if a company does not have the time to write its own Functional Specification, (or RFP), and feels that it must hire a consulting firm to give birth to a 250-page document, that company has no business trying to implement any system that arises from that document, because it will not be "their" system. That system stands, in my opinion, a better than 90% chance of failure.

The Project Manager must plan, organize, and control (in other words, Manage) the day-to-day efforts of the project. He must continually check to make sure that detailed designs will meet the needs of his (and others') departments. He must monitor progress to schedule, and adjust the schedule to the realities that intrude on the best plans. He must control requests for changes by sitting on most of them, and presenting the few worthy ones to the Steering Committee. He must chair the Project team at its weekly meetings, and the Steering Committee at its bi-weekly meetings. As mentioned earlier, the Project Manager is also the head trainer, and trains either user personnel directly, or their managers who in turn train their subordinates).

The Project Team is comprised of the Project Manager (Chairman), MIS Manager (Assistant Chairman), managers of all departments affected, and the analysts and programmers assigned to the project. It is responsible for resolving differences of opinion that do not involve policy or fundamental operating philosophies, recommending policy and operations changes to the Steering Committee, ensuring that the project progresses as scheduled and results in the benefits promised, and prioritizing the myriad requests for changes, modifications, enhancements, etc. that occur in such projects. It must also monitor the creation and installation of internal controls, and contingency plans.

To be effective, the Project Team needs to meet weekly (a standing meeting time and place is usually appropriate). They need to keep a formal "Problem List," with the status of each problem, including its final resolution and date. This will ensure that a problem does not get ignored until it becomes extremely costly to resolve. The Project Team must send minutes of its meetings to the Steering Committee, with the Outstanding Problem sheet attached, annotated to show how each problem will be resolved. Finally, the members of the Project Team must be the ones who train on the new system first, and best. They will be assisting their subordinates to use the system correctly; they need to understand well how it works. They also need to know the inner workings of the new system so that the many decisions that must be made during an implementation will be the best possible.

The Steering Committee is comprised of the Project Manager (Chairman), MIS Manager (Assistant Chairman), the top executive of each department affected ("mahogany row," if you will), and the person to whom those executives report (the "corner office"). This committee should meet bi-weekly (more frequently during a "crunch"), to monitor progress, set policy,

commit resources as needed, resolve any differences of opinion that could not be resolved by the Project Team, and approve/disapprove Project Team recommendations. It should not get involved in the day-to-day implementation effort; that is why the Project Team exists. The Steering Committee must also ensure that adequate contingency plans, internal controls, and documentation exist as the system is being designed and installed.

CONTROLLING

There can be no control without adequate plans, for one must control to a predefined goal. There can be no controls without proper organization, for there would be no person held responsible. Given, however, that plans and organization exist, control is absolutely mandatory. Without control, there is no feedback to inform management of deviations from plan to allow them to redirect the implementation efforts appropriately, or to measure the performance of the persons involved. Of the three management functions, controlling is the most difficult, and the one that is least well executed, in my experience. More implementation efforts fail from lack of adequate control than from the other two functions combined.

We have discussed earlier that the Project Manager, and Project Team, must control the project on a daily basis. They must monitor progress against each of the major requirements:

1. *Time*. To do this, each project must be subdivided into tasks so small that each of them takes one person no longer than two weeks. Each of these tasks needs to be identified on a PERT chart, staffed, and tracked. This avoids the surprise of learning, one week before scheduled conversion, that the project is six months late. Progress must be reviewed weekly.
2. *Budget*. The easiest way to monitor this is to use project control software. Expenditures must be reviewed weekly, in concert with progress and projected completion dates.
3. *Benefits*. These need to be followed also, for if they are not going to be achieved, the project should be considered for immediate discontinuation by the Project Team and Steering Committee.
4. *System Performance*. Same as Benefits. If the system will not perform as expected, implementation should be stopped unless reapproved by the Steering Committee.
5. Internal Controls, and Contingency Plans. In the euphoria of system development, nobody wants to think about such things. They are absolutely essential. Internal controls can, and do, highlight system deficiencies. After our new Order Entry system had been installed for a year, our Controller insisted on installing another simple internal control. It revealed that on a very few occasions, we were not invoicing our customers for goods shipped! Contingency plans are required, because the hardware will eventually fail. (Murphy was correct; ours failed during our monthly close.) We are still in business because we had developed contingency plans.

Let me reemphasize that the Project Manager and the Project Team need to continually keep the project boundaries in firm focus. I suspect that more projects have floundered and finally sunk from the mid-stream addition of features, enhancements, etc. than from any other single cause. Once the Functional Specification is approved, there should be no major changes without Steering Committee approval. Once the Detail Design is finalized, there should be few if any changes allowed.

If a package is being installed, requests for change should be segregated into three categories: a) Must Have Before Implementation, b) Should Have As Soon As Possible, and c) Nice To Have. There should be very, very few changes in category a); these are the ONLY changes that should be permitted before implementation of the standard package. Once the package is installed and running, over half the requests in categories b) and c) will disappear; they will no longer be necessary. Each change that is permitted to delay the installation of the package delays the benefits that will be derived from installation, and increases future maintenance problems.

The Steering Committee must measure progress against plan for all major dimensions outlined above, and ensure timely completion to specification. They must resist the overwhelming urge to modify, or enhance, unless the benefits are extremely attractive. They must be willing to scrap the project if the costs grow, as is usually the case, and the benefits shrink, as is also usually the case, to the point at which it is no longer financially attractive, as is fortunately the case only occasionally. Finally, they must ensure that old systems are left intact until the new system has proven that it really works. I visited a company some years ago that demonstrated the validity of this last point. They had destroyed the old system; the new one had not worked for two months. The people in the plant were playing cards.

CONCLUSIONS

Systems do not implement themselves; people implement them. To succeed, a systems implementation effort must be managed effectively, by applying standard management principles (Planning, Organizing, and Controlling) with the intent to minimize risk. This is accomplished by using a time-phased commitment approach that provides management three separate opportunities to review costs and benefits and schedules, and to discontinue the effort with only the minimum possible resources having been expended at each of those decision points.

It is imperative that we, as MIS professionals, cause systems to be implemented properly in our respective companies. Our companies cannot afford the disaster of systems implementation failure. We cannot afford the continued negative publicity, and the resultant scepticism concerning our professional competence. Or, to be more blunt, a manager is only as good as his ability to deliver on his promises; we have proved for 20 years that we still lack that ability. It is time for us to acquire it, or face the consequences.

BIBLIOGRAPHY

Bliss & Laughlin Industries. Corporate Data Processsing Standards Manual. Oakbrook, Illinois.

Edson, Norris W., "The Realities of Implementing MRP," 23rd Annual Conference Proceedings (1980), American Production and Inventory Control Society, Inc., Washington, D. C.

Jones, Gary D., "Pitfalls to Avoid in Implementing MRP," 21st Annual Conference Proceedings (1978), American Production and Inventory Control Society, Inc., Washington, D. C.

Lasden, Martin, "Turning Reluctant Users On To Change," Computer Decisions, January 1891, pages 92-100.

McFarlan, F. Warren, "Portfolio Approach to Information Systems," Harvard Business Review, September/October 1981, pages 142-150.

Olsen, Robert E., "MRP Implementation — Doing It The User Way," 21st Annual Conference Proceedings (1978), American Production and Inventory Control Society, Inc., Washington, D. C.

Orr, Kenneth T., "Systems Methodologies for the 80s," Infosystems, June 1981, pages 78-80.

Salmere, Mitchel B., "How to Improve a Management Information System," Infosystems, November 1981, page 90.

Wight, Oliver. The Executive's New Computer, Reston Publishing Company, Reston, VA, 1972

APPENDIX A1

The Functional Specifications can include any and all of the following topics:

- General Background
- Management Summary
- Problem Definition
- Present System Description
- Major System Objectives
- Proposed System Description
- Economic Justification
- Detailed Plan of Action
- Responsibilities
- Proposed Schedule

APPENDIX A2

Functional Specifications for a system can include the following topics:

- Major Logic Chart(s)
- System Narrative
- Design Notes and Concepts
- Proposed Input and Output Layouts
- Proposed Controls
- Anticipated Throughput Volumes
- Future Capabilities
- Environmental Constraints on Expansion Capabilities
- Hardware and Software Considerations
- Proposed Training Plan
- Cost Considerations and Assumptions
- Interface Considerations
- Audit Considerations
- Contingency Plan

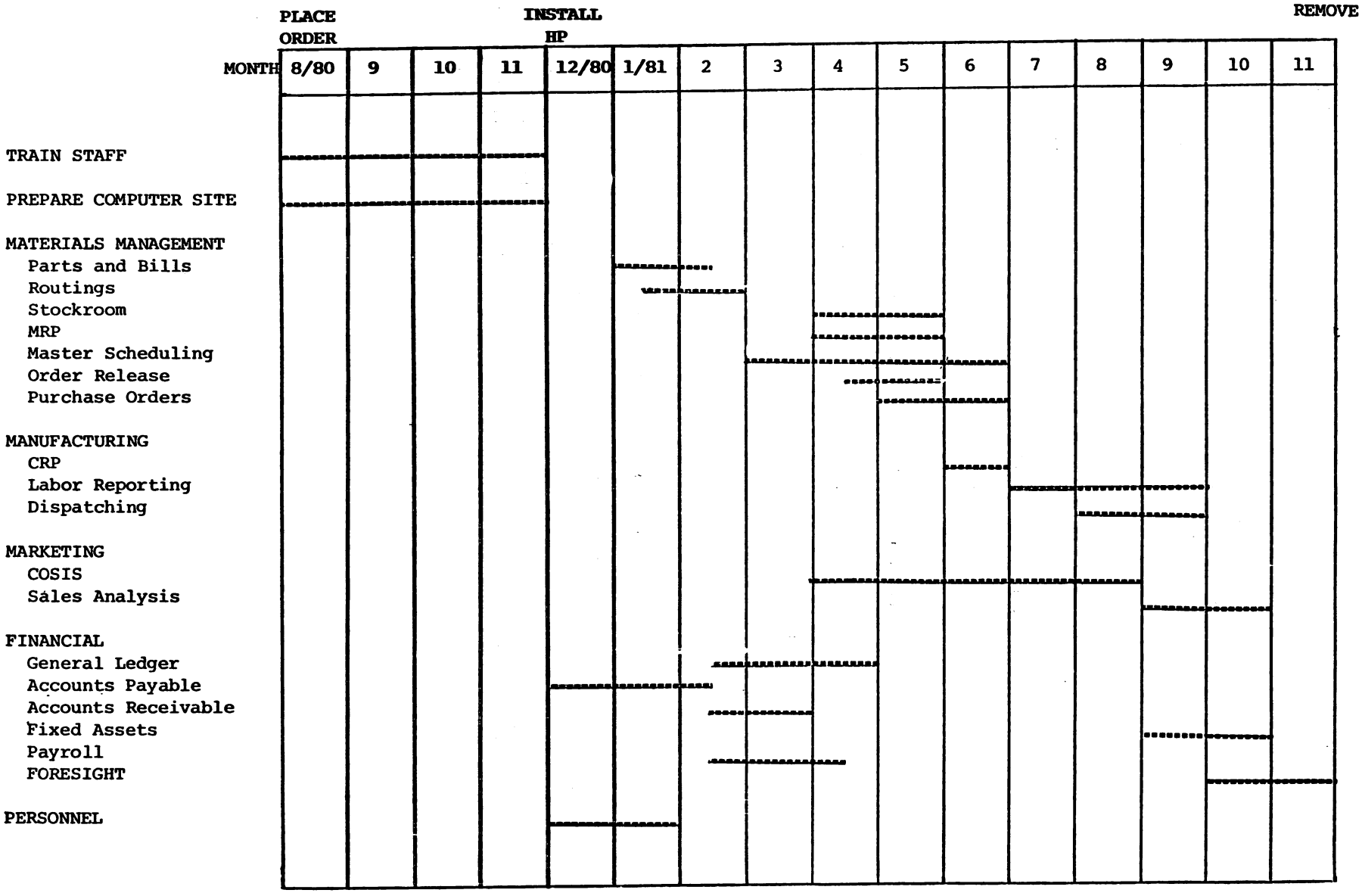
APPENDIX A3

These items should be included in a Detail Design Specification; others may be added at your discretion:

- Detailed System Flowchart, defining input and output for each program
- Detailed Narrative for each section of the flowchart
- Program Run Sequences
- Audit Measures
- Quality Control Measures
- Internal Control Measures
- Security
- File and Data Conversion from the Present System
- Recovery Procedures
- Programming Conventions
- Test Specifications
- Test Standards
- Hardware/Software Environment
- Program Narratives
- Program Functions
- Program Specifications
- Program Logic
- Detailed Project Implementation Schedules

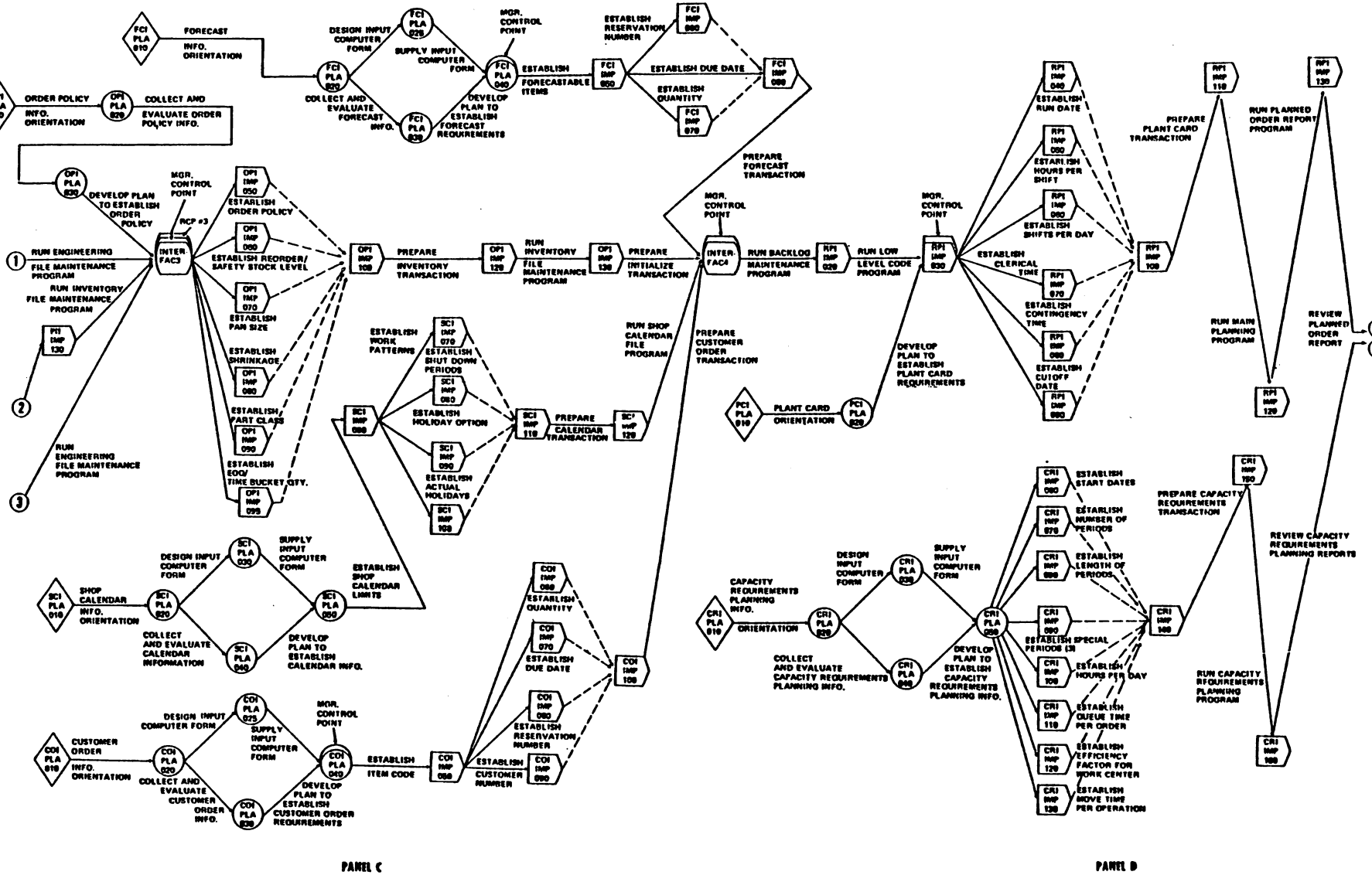
SYSTEM IMPLEMENTATION MILESTONE CHART

(Assumes BLI approval not later than 8/1/80)



APPENDIX B
BAR CHART

APPENDIX B
PERT CHART



An Overview — Networking Cost Performance Issues

Russell A. Straayer
President, Data Communications Brokers, Inc.
Champaign, Illinois

The purpose of this paper is to give managers a conceptual overview of several key issues in datacommunications networking. We will focus on several practical, useful guidelines. At the end we will address what is practical and sensible today for most applications. As managers, you cannot wait for the promises of 10 years or even 5 years from now. A basic principle we will stress is response time for the terminal user.

Response time will be stressed because the major function of an HP3000 is to serve a human being, a person using the computer through a computer terminal. As human beings, we demand fast response times. We take our cars instead of waiting for the bus. We take the plane because the train or car is too slow for long distances. We eat at fast food restaurants. We read newspapers because we can scan vast amounts of information, pages and pages, in just seconds. We are equally demanding of fast response and convenience from our computers.

Getting to some practical information we can use today, we will first look at 10 specific topics, and cover them in rapid fire order. The 10 items could be full blown topics in their own right. What we as managers want to get out of the 10 points, however, are the practical guidelines. The technical staff folks and the vendors all have their technical pitches, and we must distill out sensible solutions that work today.

The 10 points, then are:

1. Telephone line cost trends
2. Data communications hardware cost trends
3. Technology trends of datacommunications hardware
4. Local networking alternatives
5. Packet protocol caveats
6. Satellite communications caveats
7. Importance of fractional second response time
8. Bits per second versus speed
9. Point to point versus multipoint cost/performance considerations
10. One large, efficient network

We will look at each one of these 10 points and highlight the important points using a few charts, graphs and illustrations. After we get through the 10 points, we will have a better basis for quickly getting to some firm network approaches. While there are many choices, many of them correct, most fall within a narrow range of practical solutions. In the end, it is then the responsibility of the manager to choose a solution that works. There will be no right or wrong decisions, just some that fit better than others. The ones that fit the best will be those made by management that has a good idea of the direction in which it is going and the destination it wishes to reach, and then asks the right questions to get most directly to the destination.

POINT 1. *The trends in telephone company communications line costs.* The trend in costs is up. Costs for lines are increasing at about 16 percent annually. We have an example of Illinois local private line rates. In 1970, a local private line cost just $3.50 per month, went to $15.00 by 1975 and is now up to $35.00 per month. That is for a line that may just go across a street. Telephone company charges for toll calls have not gone up much at all in the same time frame. Take note, however, that the phone company is slowly but surely working to put even local calls on a usage charge basis. The local call usage charges are already in place in Chicago, New York, Washington and other cities, both in the United States and elsewhere in the world.

The upward price pressure is clearly on dedicated facilities. Today, many areas in the United States have not yet caught up to Illinois, but will. You can look upon the Illinois example as a benchmark for where rates will go throughout the United States. The telephone companies are going to the state public utility commissions and gradually raising these private line rates.

Guidelines we can draw from this are:

1. Economize on lines using statistical multiplexers
2. Economize on lines using split stream modems
3. Be aware that local dial up lines can become much more expensive
4. Satellite and private local facilities may not yet be less expensive than telephone company private lines.

POINT 2. *Datacommunications hardware trends.* Users have been spared the full impact of the rising phone line rates by the reduction in the prices of datacommunications hardware. You are all probably familiar with the constant reduction in the prices of CRT displays and printing terminals. The prices for modems and multilexers, two key datacomm ingredients for on

line networks, have also seen prices come down. In 1970, a 9600 bits per second modem cost about $10,000, or "a buck a bit." Today, some 9600 bps modems can be purchased for as little as $3,000. The prices of 4800 bps modems have dropped from $5,000 in 1970 to half or less than half of that price. Statistical multiplexers, a product just about 3 years old now, has seen a 25% price reduction in its young product life.

Guidelines we can draw from the hardware trends are:

1. Expect these approximate hardware costs;
   - 0 to 300 bps dial up modems now cost $200 to $300 per unit,
   - 1200 bps full duplex dial up modems, $700 to $900 per unit,
   - 2400 bps synchronous 201 type modems, $700 to $1,200 per unit,
   - 4800 bps sync 208 type modems, $2,000 to $4,000 per unit,
   - 9600 bps sync 209 or V.29 modems, $3,000 to $6,000 per unit,
   - 4 channel statistical multiplexers, $1,500 per unit,
   - 8 channel statistical multiplexers, $2,400 per unit,
   - Short haul synchronous modems, $600 per unit,
   - Short haul asynchronous modems, $300 per unit,
2. Expect prices to come down, features to be added, or both.

POINT 3. *Technology trends.* The technology trends of datacommunications hardware have also been at work to spare the user the full impact of rising phone line rates. The statistical multiplexer just mentioned is a perfect example. The stat mux, as it is called, has improved the price performance of ASCII CRT's and printers by more than a factor of two. The stat mux makes more efficient use of the phone line, lets the async ASCII terminal run faster than it could before, and often at lower costs than some slower, less efficient methods.

Another technology improvement of just a few years ago that we already take for granted is the 1200 bps full duplex dial up modem, equivalent to the Bell model 212. The 212 has been around for just about a half dozen years. Look for 2400 bps full duplex dial up equipment, at an affordable price, in the very near future.

The technology in datacommunications is making more efficient use of existing facilities, facilities are getting faster and more reliable, we are getting more control over the equipment, and more features. Microprocessors are showing up in more datacommunications equipment every day. We can do more and more for less and less. Just look at how vendors are scrambling to stay profitable in the face of this trend by giving you more and more features for the same prices, or even lower prices. That is good news for all users.

It is useful to look at the changes in technology over the years 1970, 1975, today, and what we may see in 1985.

1970
- Frequency division multiplexers (FDM)
- Time division multiplexers
- 103.113 type 300 bps modems
- 202 type 1200 bps modems
- 201 type 2000 to 2400 bps modems
- 208 type 4800 bps modems

1975
- Same as 1970 plus
- 212 type 300 to 1200 bps modems
- 209 type 9600 bps modems
- Short haul modems
- Coaxial cable modems
- First diagnostic tools

1980
- Same as 1975 plus
- Statistical multiplexers
- Integrated technical control systems
- Satellite
- Value Added Networks (VAN)

1985
- More software in datacomm products
- More features
- More cost effective local network products
- More cost effective fiber optics
- 2400 bps full duplex dial up modems

Guidelines drawn from technology trends:

1. Expect hardware to be very reliable, 20,000 to 50,000 hours Mean Time Between Failures (MTBF)
2. Look for simple to use features. For example, test functions that are useful but not too detailed if you or your staff do not use the functions daily. What good is it to know your bit error rate is 1 in 10 to the 6th, if you are not sure if that is good or bad.
3. As time goes on, look for more features for you money.

POINT 4. *Local networking.* Before we get to some of the details about local networking, take note that many local network products are still more promise than reality. Most installations today cannot yet take advantage of local networking technology because of the costs, or lack of interface compatibility.

Local networking alternatives include protocol options such as ethernet collision/detection, token passing methods, time division or frequency division. Links available include coaxial cable, twisted pair, infrared, microwave, and fiber optics. Hardware includes short haul modems, coaxial cable modems, ethernet type of interfaces, PABX's with data channel capability, and more hardware appears with increasing regularity. A good local networking overview article can be found in the December, 1981 issue of Datacommunications Magazine.

Guidelines concerning local networking:

1. For the HP3000, short haul modems are the most practical local network product.

2. Fiber optics are usually too expensive.
3. Ethernet type solutions are usually too expensive yet today.
4. Coaxial modems are often more expensive that short haul modems.
5. The short haul modem solution is practical today only within several miles of the computer. Beyond that, long haul modems are usually required.

POINT 5. *Packet Protocol Caveats:* Packet protocols that you hear about today include X.25, HDLC, SDLC, and many of the protocols found in the Value Added Network services such as Telenet and Tymnet. These protocols have a place, but are not going to be a complete solution. These protocols are:

- Designed mainly for message transfer, packets, electronic mail
- Too slow for full duplex operation
- Not suitable for the HP ENQ/ACK terminal to CPU handshake

As a practical guideline:

1. The packet protocols may have a place for mainframe to mainframe communications.
2. The packet protocols are rarely practical for terminal to mainframe communications.

POINT 6. *Satellite Communications Caveats.* The basic fact to consider abvout satellites is that they sit in a stationary orbit about 25,000 miles above the earth. It is that physical fact that contributes to the plus and minus features of satellites. First of all, satellite costs are not mileage sensitive. It matters not whether you go across the street or across the country, the cost is the same. Keep in mind that because it is not mileage sensitive, satellite is cost effective only on links of 500 miles or more.

A major consideration for satellite is local distribution of the data once you get it off of the satellite. From one major point to another major point, satellites can be cost effective, but not from many diverse points to many other diverse points.

Satellites are not very good for highly interactive data. Any transmission using satellite is bound by the speed of light, 186,000 miles per second. A round trip for data, via satellite, is 100,000 miles. Figure the round trip time to be almost 3/4 second.

Guidelines for satellite transmission:

1. Because of the round trip delay time, satellite is not suitable for the HP ENQ/ACK handshake or polled terminals (MTS).
2. You may wish to use satellite to connect mainframes, but not to connect terminals on line.

POINT 7. *Importance of fractional second response time.* At first glance, it may seem improbable that people need a 1/4 second response time or less to be efficient in interactive applications. It is in fact rather difficult to really grasp just how long 1/4 is.

To illustrate the importance of such small portions of time, consider the example of a radio call in show. We have all heard the person who calls in, and while listening to himself on a background radio, gets confused. The announcer says, "Please turn your radio down." Callers hear their own voices, fed back over the radio, but the delay disorients and confuses the caller.

To site another example, a user typing at a terminal will become very inefficient if the typed characters echoed back are delayed by just 1/4 second. The terminal user types ahead of the display rate of the characters and experiences what feels like a spongy keyboard. When a mistake is made at the keyboard, extra characters must be erased and retyped just to get back to the incorrect character.

When users are on line, say in a telephone order situation, it is important, if the person is to work at maximum efficiency, to get feedback for each keypress in 1/4 second or less. Typing at just 45 to 50 words per minute requires a key press every 1/4 second. When the keys do not get echoed back by the computer within that 1/4 second time window, the user slows down to match the echo time.

Keep in mind at this point that a high bits-per-second rate does not automatically mean fast response time. This fact can be easily illustrated by considering a 300 megabyte disk that we send by mail. If the post office delivers the disk in just 3 days, the transfer rate equals 9600 bits per second.

Guidelines for considering response time:

1. Response time over the communications link should be measured in milliseconds for interactive use. If the user operating a local terminal sees no delay, then the remote terminal user should see no delay, either.
2. For batch work, for file transfers and electronic mail, response time is less important than the bits-per-second transfer rate.

POINT 8. *Bits per second versus speed.* It may seem contradictory at first, but 9600 bits per second may not be as fast as 2400 bits per second. The difference is response time difference.

A Bell 2400 bps modem, model 201 allows for much faster polling in an MTS environment than does a 9600 bps modem, model 209. The difference in response time is accounted for in the Request To Send/Clear To Send delay functions of these modems. In a multipoint polled (MTS) environment, the RTS/CTS delay allows the modem at the central computer site to "tune" itself to the incoming signals from modems at any one of several remote sites.

The 2400 bps modem has an RTS/CTS delay of only 7 milliseconds, while the 9600 bps modem has a delay of 147 milliseconds. Given a typical poll of 12 characters and a 3 character response (a total of 15 characters), the

2400 bps modems allow for 3 times as many polls per second. For short message traffic, the lower speed modem may be a good bit faster.

Guidelines regards bits-per-second versus speed:

1. In a polled (MTS) environment, 2400 or 4800 bps is often the best you can do for your money.
2. File transfers to an RJE station or mainframe to mainframe can benefit from the 9600 bps modems, since there is usually no concern about the RTS/CTS delay.
3. Using statistical multiplexers and asynchronous terminals, the typical best speed is 2400 bps for the terminals, and 4800 bps for the composite modem link, for up to 8 terminals. 9600 bps may be called for if printers are heavily used.

POINT 9. *Point to point versus multipoint (MTS).* In an HP3000 environment, point to point usually means using asynchronous CRT's and printers. Multipoint is the MTS environment. What we want to do here is look at the differences from the datacomm point of view.

We have a case study to look at comparing MTS with point to point using statistical multiplexers. The user is in Dundee, Michigan. The test involved 2 CRT's and 1 printer, running on 4800 bps Bell 208 modems. The specific test was to evaluate how the user saw response time, and to measure actual output volumes.

The results were clearly in favor of the statistical multiplexer method of operation, even when terminals were slowed from 4800 under MTS to 2400 bps async. Measured output was more than double for the async mode of operation. The users at terminals saw noticeable reduction in response time when the printer was running, but not when using the asynchronous mode and statistical multiplexers.

It should be noted, too, that the async terminal operation is usually easier to set up, easier to diagnose, easier to maintain.

Guidelines on point to point versus multipoint:

1. In most cases on an HP3000, point to point asynchronous operation proves to be the most cost and performance effective.
2. You may wish to consider MTS (polled terminals) if you do very little printing along with CRT displays at remote sites and your response time can reduced by a few seconds. The printer is the major consideration.

POINT 10. *One large efficient network.* The network we will examine here is the United States switched phone network, the one we use when dialing local or long distance calls. The US phone network, managed primarily by AT&T, is well developed, efficient, and employs the princples of good networking.

Examining the routing of a phone call from Champaign, Illinois to San Antonio provides a good look at the structure of the network. A typical call is routed:

1. From the local telephone station, over a station loop to the central or end office.
2. The end office connects to a toll office via a toll connecting trunk.
3. The toll office, say in Champaign, connects to another toll office via an intertoll trunk. There are several classes of toll offices in the network hierarchy.
4. To avoid going through the entire chain of toll office command, the call may be routed from one lower level office to another, close to the destination. The lower level offices are connected via high useage trunks (HUT's).
5. The toll office close to or actually in San Antonio connects the call to the end office in the city, which rings the local phone.
6. The call is completed when you pick up the phone.

This entire process is referred to as circuit switching, since the call connection uses actual, physical circuits. Packet switching, on the other hand, does not make a connection via actual circuits, but packages up the data and routes the databased on destination addresses included in the packages.

Suggested guidelines based on the phone network:

1. Think of your HP3000 as though it is a PABX on location in a business. Its purpose is to connect terminals to files and terminals to terminals.
2. Terminals will almost always be connected to the HP3000 the way phones are connected to a PABX or central office, with one port per terminal, just as there is one physical line per phone number or extension.
3. Access through the HP3000 should be as standard and simple as possible.

At this point, we will look at a specific network on an HP3000. The user here is Johnson and Staley of Nashville, Tenessee. This network takes all this information and illustrates the kind of datacommunications most practical for 90% of all HP3000's.

The Johnson and Staley application is on line order entry and inventory maintenence for a distributer of school supplies.

The Johnson and Staley application embodies our 10 points in the following ways:

1. The line configuration is designed to keep the phone line costs to a minimum.
2. The links between the multiplexers are 4800 bps, about the best in price for the bits-per-second rate needed. Five years ago Johnson and Staley would probably have decided that the modem and multiplexer costs were to high to go on line.
3. The DDS, bandsplitting of DDS, and the stat mux's are all late 1970's technology.
4. Local networking is not applicable here.

5. Satellite links are not cost or performance effective here.
6. Packet networks are not cost or performance effective here.
7. The on line order entry activity required very fast response times.
8. A link of 9600 bps per terminal grouping would not improve performance in this application. The volume of data is small for this application. The need is for instant access to the inventory and order records.
9. Line cost savings that might otherwise be available only thru multidrop networking are acheived by bandsplitting.
10. Terminals are connected on a per port basis to the HP3000, much like extensions to a switchboard.

SUMMARY OF HOW 90% OF ALL HP3000'S COMMUNICATE

HP3000 to HP3000 or larger mainframe:

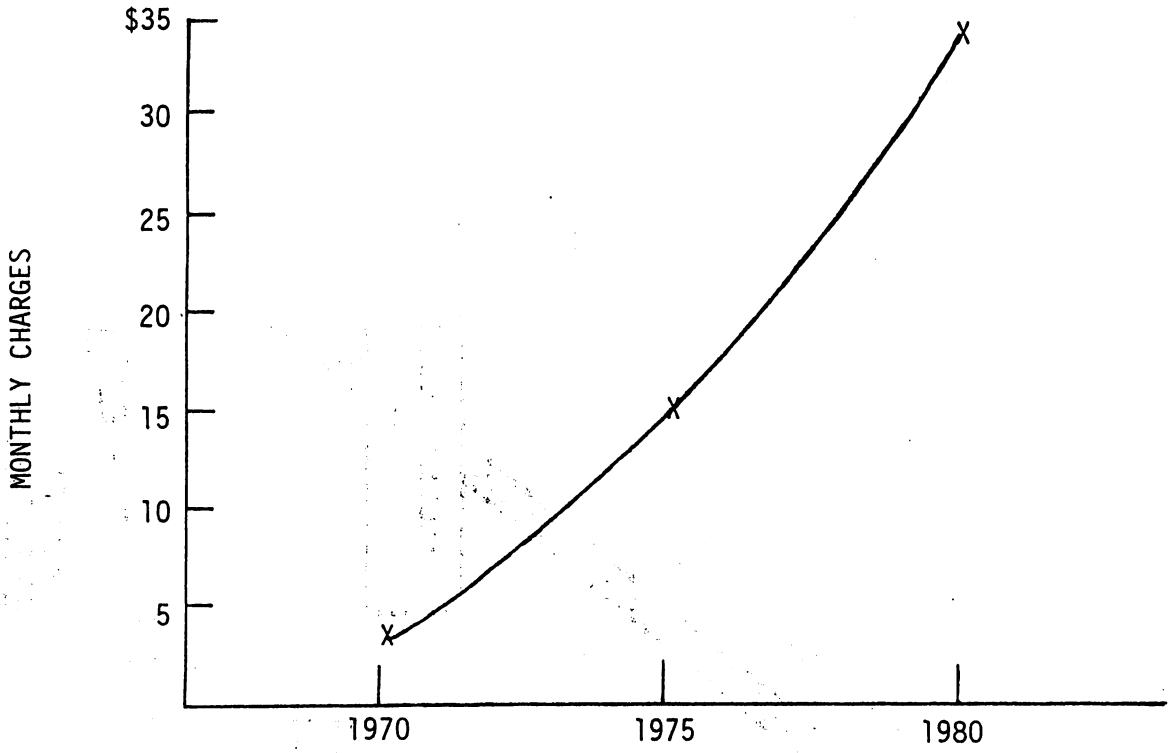
- Synchronous facilities
- Private line
- 2400, 4800 or 9600 BPS
- Digital Data Service (DDS)
- Very few satellite links

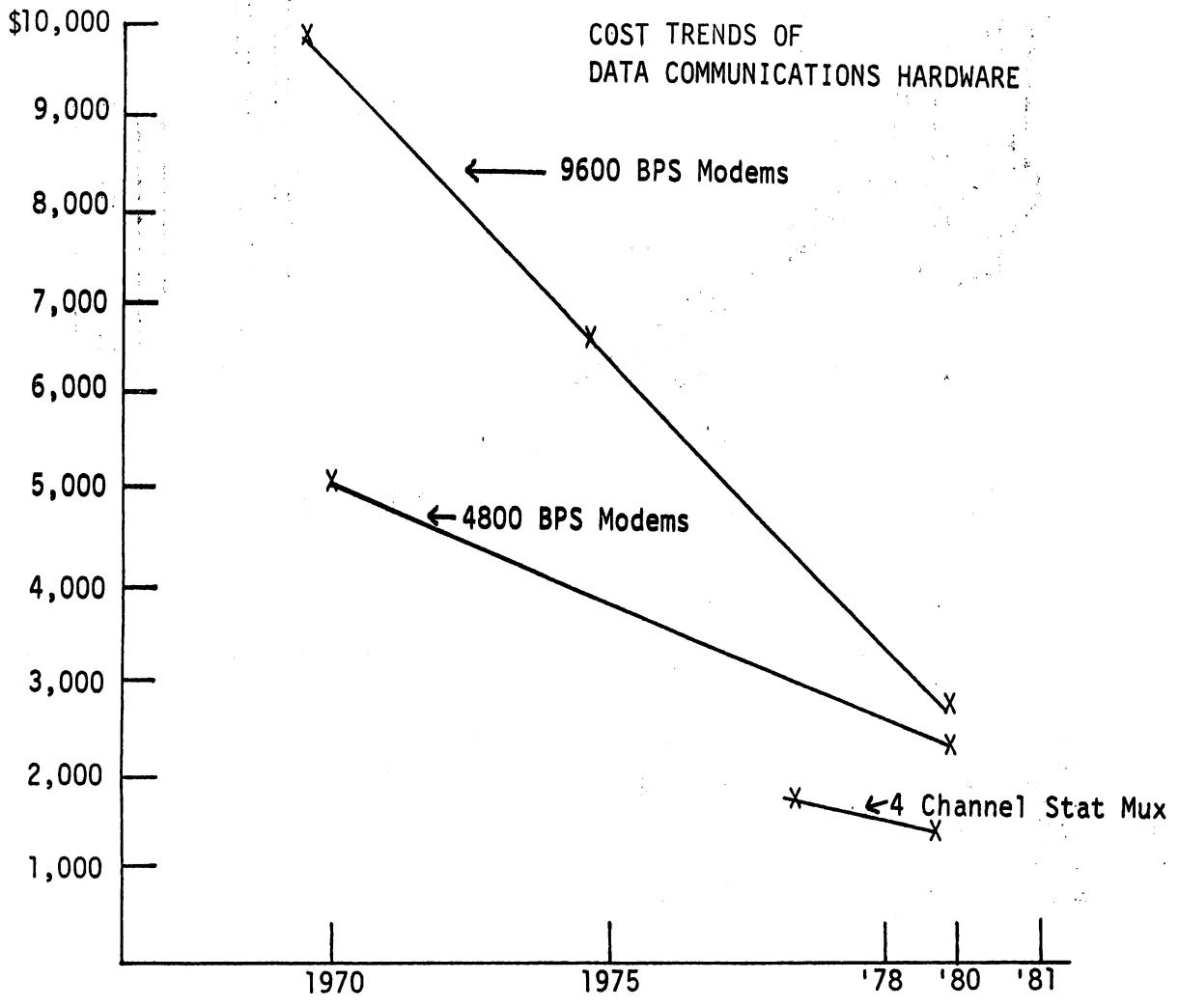
HP3000 to terminals:

- Hardwired or within 100 miles of the mainframe
- Asynchronous, with dial up, single modems or stat mux's
- Speeds of 1200 or 2400 BPS

In conclusion, for all the choices and possible confusion surrounding HP3000's in a data communications environment, 90% of all systems have the same configurations, with minor variations.

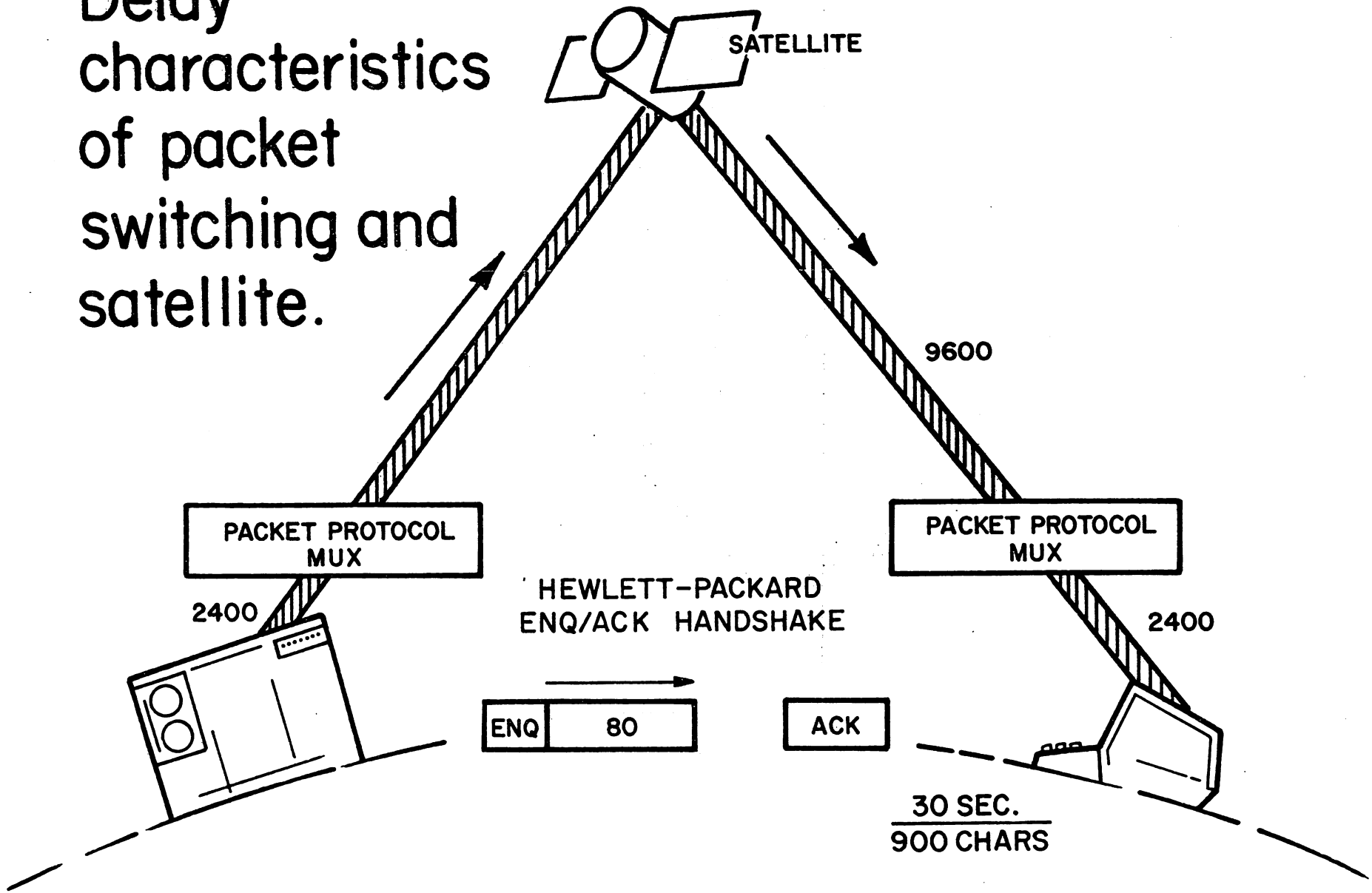
LOCAL PRIVATE PHONE LINE RATES
ILLINOIS



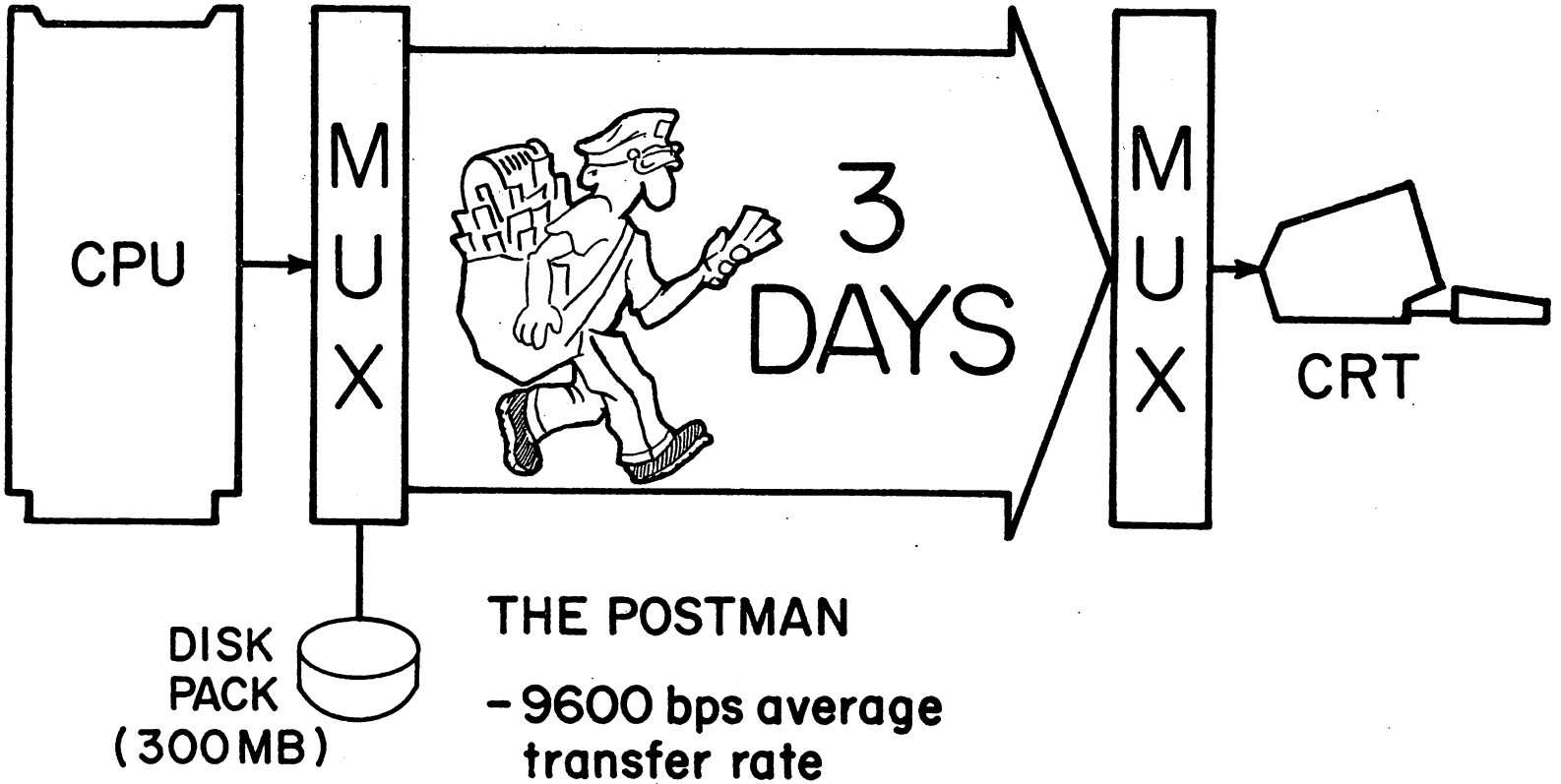


C.C.I.81

Delay characteristics of packet switching and satellite.



The Postman delivers 9600 bps.



THE POSTMAN

- 9600 bps average transfer rate
- Good for large volume several day delivery

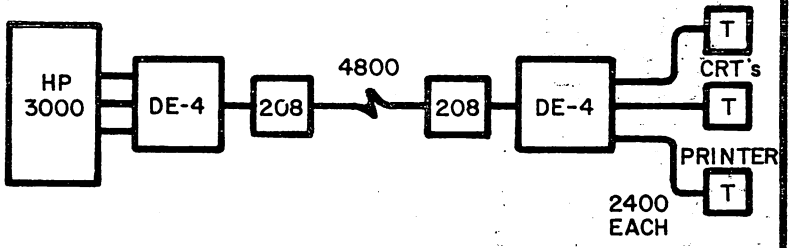
C.C.I. 81

BENCHMARKED 12-79

2 network alternatives

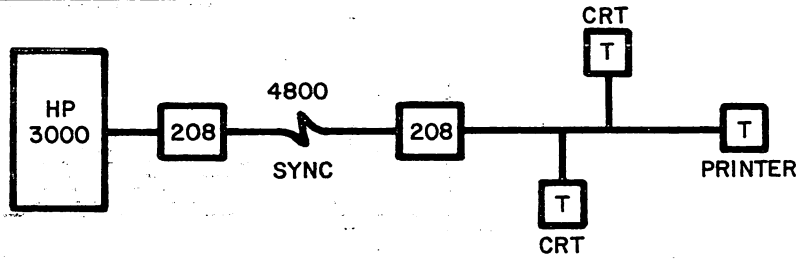
Asynchronous stat muxed terminals

POINT TO POINT



Polled

MULTIDROP/MULTIPOINT



TESTED AT DUNDEE CEMENT, DUNDEE, MICHIGAN

Total one way output...

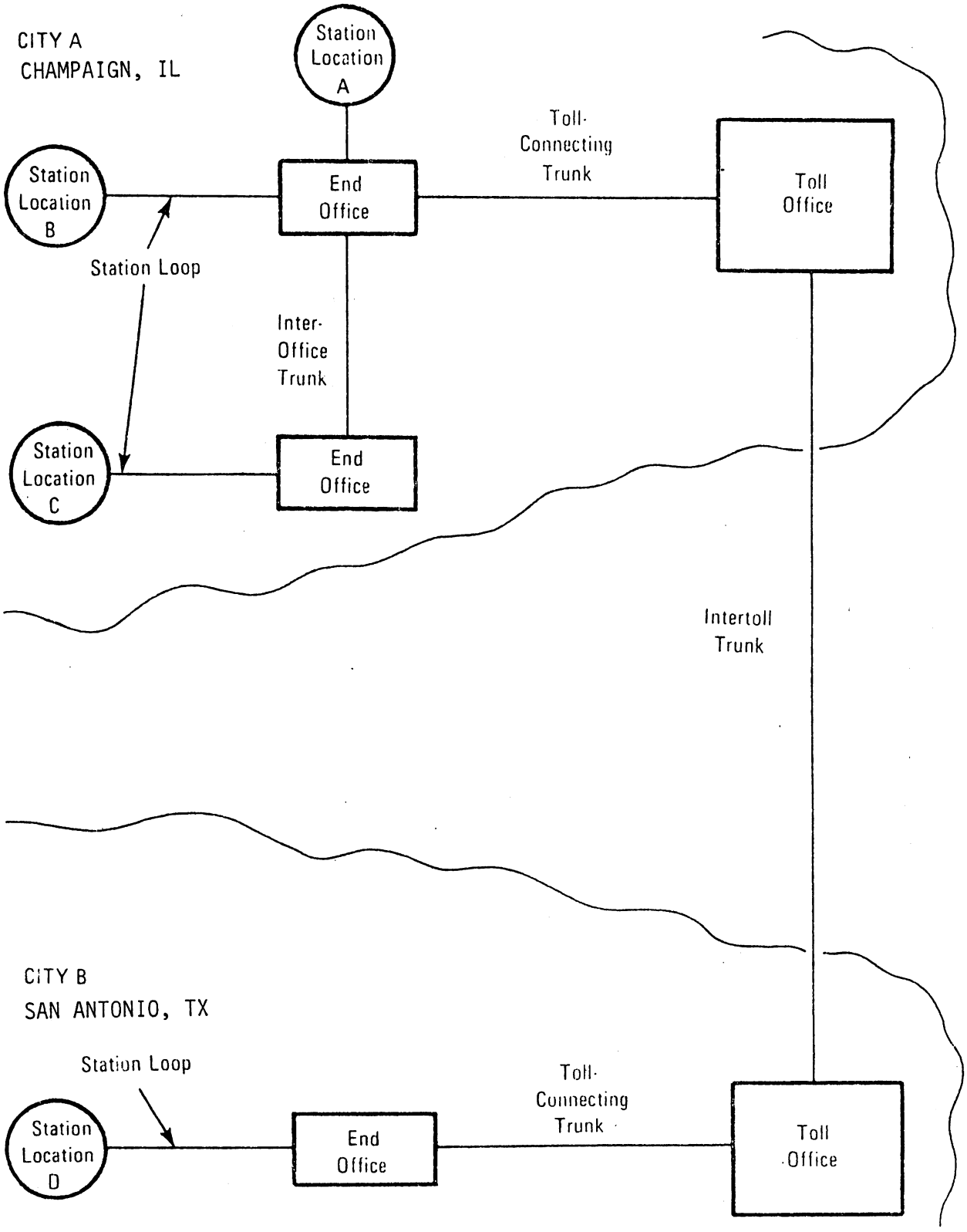
listing only —
280 CPS Average

Same modems, terminals, data output – ASYNC Interactive

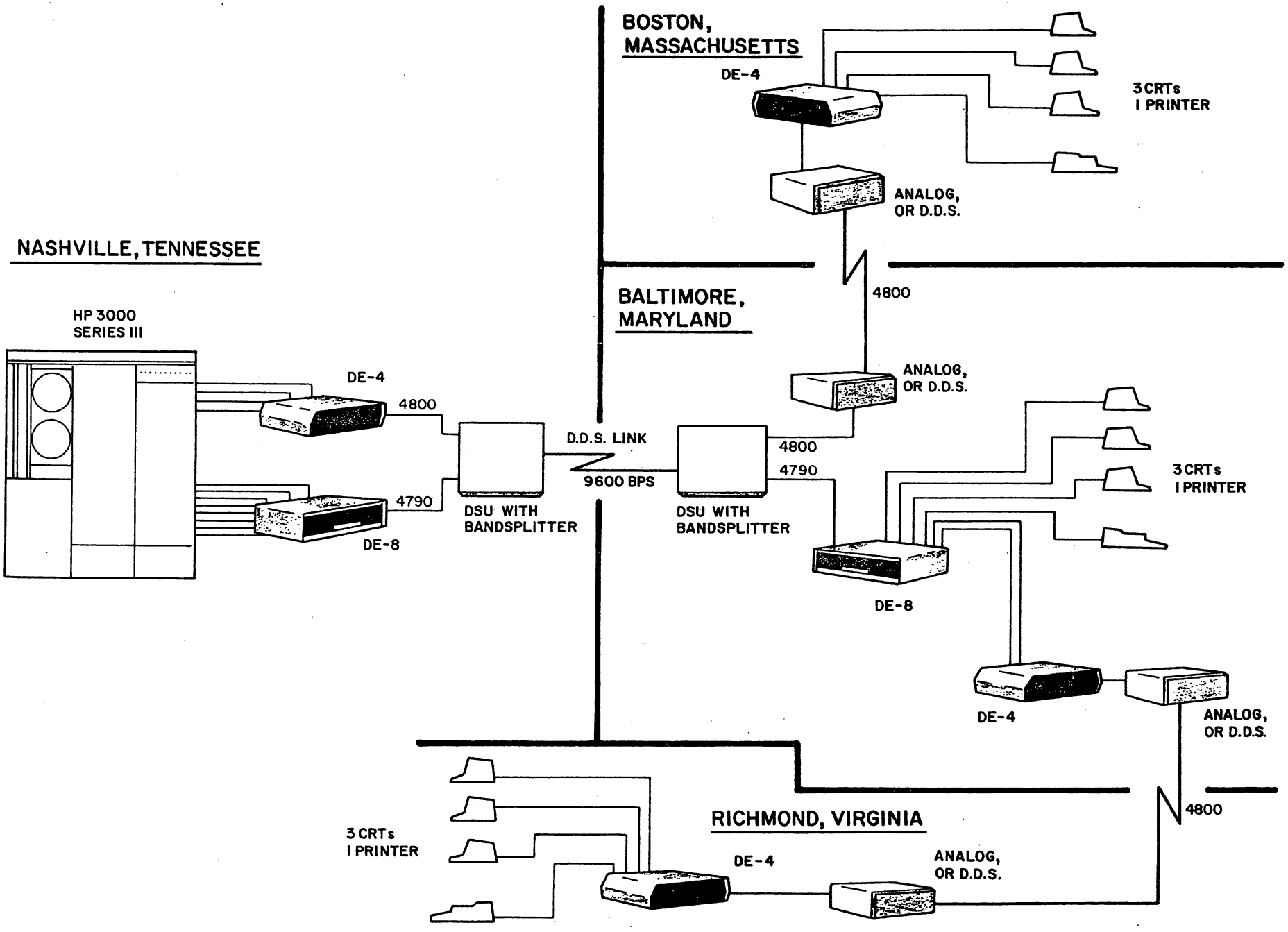
Total one way output...

listing only —
115 CPS Average

Same modems, terminals, data output – SYNCHRONOUS Polled



Typical Routing for Connections


BOSTON,
MASSACHUSETTS
DE-4
3 CRTs
1 PRINTER
ANALOG,
OR D.D.S.
NASHVILLE, TENNESSEE
4800
BALTIMORE,
MARYLAND
HP 3000
SERIES III
DE-4
4800
ANALOG,
OR D.D.S.
D.D.S. LINK
9600 BPS
4800
4790
3 CRTs
1 PRINTER
4790
DSU WITH
BANDSPLITTER
DSU WITH
BANDSPLITTER
DE-8
DE-8
DE-4
ANALOG,
OR D.D.S.
4800
RICHMOND, VIRGINIA
3 CRTs
1 PRINTER
DE-4
ANALOG,
OR D.D.S.

Microcomputer-Based Distributed Processing

John J. Tibbetts
Vice President, Research & Development
The DATALEX Company

INTRODUCTION

If one were to attempt to list the major technological changes of the last decade, surely at the top of that list would be the so-called microcomputer revolution. Over the last 10 years intelligent devices have jumped from the research and development labs out into the hands of scientists and engineers and then into business and then into small business and the home. I believe that the depth to which microcomputers have penetrated our society would have surprised even the most adventurous of futurists of 10 years ago. Nor does it appear that this wave of change is at all slowing down. Projections show very rapid growth, both in the home and business, over the next five years.

Also interesting to observe is the evolution of the packaged microcomputer as it exists today. Ten years ago the first microprocessor chips were just coming out. A few years later, a few poorly capitalized companies originated the packaged microcomputer with a microprocessor, support chips, power supply, and a limited amount of memory. Over the next couple of years the microcomputer was really in the hands of the electronics buff and the amateur radio operator with very little emphasis being placed on software. The professional computer person of five years ago approaching a microcomputer was faced with theprospect of working with a virtually bare machine in terms of professional software tools. But in the last five years there has been a rapid growth of professional software, including commercial-grade operating systems, languages, and some applications software packages. Where a few years before the microcomputers were the province of the hardware junkies, now even the major microcomputer manufacturers had very much awakened to the role of software in selling computers.

A striking poster that is now being distributed by Apple internally as well as to its sales outlets, says in massive letters "SOFTWARE SELLS SYSTEMS."

However, despite the large number of software products that are now avialable, there are actually a relatively small number of applications represented. Pick up a BYTE magazine (that is, if you can. What started out a a pamphlet now runs 500 pages, most of which are ads) and perform the following exercise. Take a clean sheet of paper and start writing down software products, grouping them according to function. You will find an interesting pattern. There will be a large number of games. There will be a number of packages which are small business accounting packages. There will be a preponderance of word and text processors, some of which are quite good. There will be some financial modeling software in the Visicalc and Plan 80 sense, used for limited, but very interactive, financial modeling applications.

There will finally be a group of programs calling themselves database management systems. I hesitate to call true database management systems in the classical sense of the word. Some of them are certainly capable file management systems.

What seems to be almost totally missing are products which are important to the kind of people who attend these meetings, that is, people with larger computer needs. What is missing is software that emphasizes the nonpersonal use of personal computers. Let's examine what a few of these software product categories might be:

1. *Communications software.*
   Communications software accurately moves transactions or files of data back and forth from your microcomputer to your corporate computer. It is true that there are some communications packages which do exist for microcomputers, but I would assert that very few of them are oriented toward commercial grade data handling chores which should include such features as error detection and retry, bidirectional control of the communications stream from either the microcomputer or the host computer, and the ability to accommodate full binary transfers of data. These matters are discussed in much greater detail in another talk which I am giving at this meeting.
1. *Intelligent terminal software.*
   By intelligent terminal software, I mean software which can format and edit transactions of data as they are being entered and before they are submitted to your corporate data processing machine or network. Nearly all online, realtime applications which are performed onto an HP3000 are performed via dumb terminals. Consequently, all of the editing and transaction formatting needs to be done by the central computer. Not only can this be slow from a processing point of view, it can also be very slow from an apparent operator speed point of view. The use of intelligent terminals software

running on industry standard microcomputers would allow increased capability and a higher performance in interactive applications.

3. *Data entry software.*
   By data entry software I mean software similar to the intelligent terminal software just described but which can act independently of the remote computer entirely. That is, the transactions as they are gathered are stored locally — usually on a diskette on a microcomputer — and are maintained on the microcomputer until such time that all of the data has been entered. This type of software solves the standard data entry needs of data-intensive commercial applications.

In this talk, I will address myself to microcomputer software being used as intelligent terminal and data entry software and relate to you some of my experiences in this regard.

BUILDING BLOCKS

Before we can discuss the particulars of intelligent terminal and data entry software, it would be important to first define some of our terminology in terms of industry-standard microcomputers and industry-standard software for microcomputers. In this section, we will define some of the assumptions we have used in the building of our software systems.

Let us first consider hardware. Microcomputers can of course come in many sizes and shapes, all the way from the little $200 Sinclair Microcomputer on the low side to the very expensive microcomputers bordering on minicomputers on the high side. In general, I feel that the standard minimum configuration for a general purpose data entry based microcomputer to be a 64K system or greater, with floppy disk support. The reason I recommend a full 64K system for your microcomputer is simply that the cost of main memory has dropped to the point where the software costs associated with working in smaller memory sizes outweigh the amount you spend on memory unless you are producing a special-purpose, high volume data entry product. It is interesting to note that many of the microcomputer manufacturers have moved from a position of giving you whatever memory size you would like, to recommending and then strongly recommending 64K systems, and now some of them are selling only that configuratin. Although Winchester disk technology is a very exciting element in today's microcomputing, for data entry and intelligent terminal based applications it usually isn't necessary except in two specific cases: one in which you have multiple systems, perhaps more than 3 or 4 in the same location, which you would like to share the common systems software from a single Winchester drive; or two, in cases in which there are going to be keyed lookups into larger data structures which would required the performance you can get from a Winchester drive rather than a floppy disk drive. Printers are sometimes useful in certain distributed applications but in most cases don't seem to be required.

You will notice that I make no specific recommendations on hardware manufacturers. This is because I have come to the conclusion that the main reality about microcomputer hardware is that it is in a very dynamic state. What you want in microcomputer hardware for distributed processing applications is hardware that is *maintainable, reliable, and, probably, from a large-name vendor.* Beyond that no specific recommendations on my part are advisable. The fact that I am preparing this talk two months in advance of giving it leaves plenty of room for more significant announcements to be made before the talk is even presented.

Perhaps even more important than the selection of hardware for a distributed processing application is the selection of your software operating system. I say this for two reasons:

1. If the operating system is portable enough, it will alow you to change your decision about hardware during the development of your application or during different stages of its implementation.
2. The operating system has a much greater influence over the programming techniques and systems capabilities than does the hardware in which it is packaged.

Now, what kind of microcomputer operating systems can we expect to find these days? Let's perform a mental exercise. I will entitle this exercise, "Name That Operating System." OK, name this operating system:

- Runs in a Stack Environment
- "Segmented" Architecture
- Non-Von Neumann → Code Segments Separate from Data Segments
- Up to 256 Code Segments of 65K Bytes Apiece
- Process-handling
- "Intrinsic" Procedures to Implement Supervisor Calls
- Inter-linkable Languages

Do you have the name of that operating system fixed in your minds? Good. Now, let me add a few more attributes to the list.

- Runs on 8080, 8085, 8086, 8088, Z80, Z8000, 6800, 68000, 6502, LSI-11, TI-9900
- Has 70,000 licensed users
- Supports PASCAL, FORTRAN-77, BASIC (interlinkable)

Do you still have the same operating system in mind now? The operating system I have been describing has many attributes asociated with the HP3000 MPE architecture and the Burroughs architecture before it. This operating system is the UCSD p-System, so-called because it originally developed from the PASCAL language project from the University of California-San Di-

ego. It is now marketed worldwide by SofTech MIcrosystems. I consider this operating system to be the most professional of the 9-bit/16-bit microcomputer operating systems. It gives the deveoper a capability approaching the power of MPE running in a portable microcomputer environment.

The word "portability" can't be stressed enough when dealing with microcomputing. I meantioned previously that hardware is in an extremely dynamic state in the microcomputer industry. Thus, the notion of protecting your software investment which HP has always preached to their customers is extremely important in the microcomputer domain. After all, in the microcomputer domain your software investment is often many times the cost of the hardware for small hardware configurations and the importance of protecting it against the extremely volatile hardware changes we find in the microcomputer area is very important. UCSD p-System portability means that we can take compiled, running systems and move them from microcomputer to microcomputer, even running different processors, and have them immediately execute. This is true portability.

Another building block we need for distributed processing type applications is good, commercial-grade communications software. We need software that can move transactions or files of data back and forth to the HP3000 with full error detection. Our approach has been to write compatible communications programs, both on the microcomputer and the HP3000, which provide for sending checksummed packages of data and messages back and forth between the two processors. Thee is a great deal more versatility and reliability when you have interlinked programs running on both sides. These programs will run very effectively even over noisy telephone lines or in environments in which characters, such as the important DC1 character, may suddenly disappear. The programs have the ability to time-out after priods of no communications so that the error recovery can be graceful. The programs also have the ability to allow either side — that is, either the local microcomputer or the remote HP3000 — to control the communications. Thus, for instance, we have built applications in which the operator simply starts up an HP3000 UDC, the UDC starts up perhaps a COBOL transaction processor which polls the microcomputers for the filenames which they need to send and then requests that the files be sent.

The last building block needed for distributed processing systems is a comprehensive forms language. To date, such software has not been available on microcomputers and consequently we have spent the last couple of years building it ourselves. The forms language tends to have many of the attributes of the V/3000 approach of forms building; to wit, draw a pickture of the form in a screen editor and then proceed to specify attributes, such as range checks, table lookups, optional fields and so forth about the form. The significant difference between a forms language that can be written on a dedicated microcomputer and one that can be written on a larger shared processor is in the greater degree of user interaction that can be accomplished on a microcomputer. On a keystroke-by-keystroke basis, the microcomputer can do instantaneous editing of the data, rather than waiting to gather up a whole block of data and then transmit it to some computer somewhere else for editing. This means that the microcomputer forms have a very high apparent speed, no matter what the speed of the remote processor.

Other form attributes that have been implemented are:

- A "dup" key. This key immediately copies the previously typed entry to the current data value.
- Function keys which can cause immediate action in the data, such as default values or clearing a field.
- Data verification in the IBM sense of retyping the data exactly the same way (just the way your key punchers have been trained to do it).

With these building blocks we have the tools needed to build very innovative and effective data processing nodes onto existing information network.

INTELLIGENT TERMINALS

Our approach to writing intelligent terminal software has been to use our forms language to build, compile, and maintain the forms on a microcomputer. The compiled forms can be stored on the host HP3000 for distribution. These are periodically distributed to the various nodes through the communications software. The applications program, say a COBOL program written on the HP3000, controls the microcomputer by sending down very simply formatted ASCII strings to the microcomputer to give it its instructions. Since there are no special control character sequences, the screen commands can be dispatched by any language, not just COBOL. They can even be dispatched by UDCs. For instance, the ASCII string ".CS" tells the remote microcomputer to clear the screen. The command ".LF PRODUCT" tells the remote microcomputer to load the form named "PRODUCT" from the floppy disk into memory. Since the forms live locally on the microcomputer, a form change command represents only 10 characters transmitted from the HP3000 to the microcomputer. Compare this with the 1,000 to 2,000 characters that are usually required to change a form on a non-intelligent computer. On a typical floppy disk system we usually can store from tens to hundreds of forms, depending on the capacity of the floppy disk. Once the application has displayed the form on the microcomputer, it can give them a simple command such as ".GF" to get the form. This command causes the microcomputer to issue a read for that form and does all of the local form editing and the microcomputer without any involvement by the host computer. The result is a very high apparent screen speed that is being controlled by the remote computer.

There is a very close parallel between the intelligent terminal command strings and the equivalent subroutine calls that one would issue from a V/3000 system. Thus, one can either read or write whole forms or individual fields or any combination of them. In addition, using intelligent terminal software, one can do some fairly intelligent operations on the screen. For instance, we can request that only modified fields on a form be sent back to the host computer with some identifier on each field. Another intelligent operation is the reformatting of the record on the fly, such that the fields themselves can be shifted in position with various constant data inserted into the transmision stream.

There are a couple of easily definable benefits from using this kind of intelligent terminal software:

1. *Performance.*
   Using an intelligent terminal improves the performance of the program on the HP3000 in that it does not need to be burdened with a lot of editing operations that can be done immediately by the local microcomputer. By the time the data is sent to the HP3000, it is as clean as local editing can provide. This system also performs very well for the operator who gets the benefits of immediate error checking on those fields that have had defined local microcomputer editing.
2. *Portability.*
   This system is very portable both with respect to the program running on the host computer — that is, the HP3000 — and to the program running on the microcomputer. For the host computer, since all of the commands for the screen operations are simple ASCII strings, the COBOL programs tend to be far more portable than COBOL programs with embedded forms control procedures. If one wished to take a COBOL program and move it to an IBM main frame, the only conversion required would be the standard conversion of any COBOL program from an HP3000 to an IBM system. On the microcomputer side, the intelligent terminal software — since it has already been defined to be very portable on microcomputers — can be running on an HP125 or on an APPLE II or any of the other microcomputers which support the p-System operating system. This means that the same application program can drive a variety of microcomputer-based intelligent terminals, depending on the preference of the system implementer or perhaps what hardware might be existing in the office that this system is running into.

One last note on the intelligent terminal software. Even with the benefits I have just described, unless a user has some particular need for portability or higher performance, my best guess is that the advantages would not be sufficient to cause someone to establish an intelligent terminal network instead of using dumb terminals. The real usefulness of this intelligent terminal software will come to light when we begin talking about offline uses of the microcomputer and especially their hybrid usages.

OFFLINE DATA ENTRY

Perhaps the single most significant application of the new microcomputer technology for users of existing information networks is doing offline data capture. This means that we can have our microcomputers sitting either in our data entry departments or in remote offices offline from our HP3000, gathering data, putting it onto floppy disks, doing local editing as previously described, perhaps performing batch balancing, perhaps generating proof listings of the data so that it can be visually verified, or rekey verified by the data entry operators, and then have the batches closed and transmitted for processing to the remote computer.

We see two major users of microcomputer-based data entry. The first is in collecting volume data typically entered by the data entry function of your information system. Let's quickly compare a microcomputer solution to the standard existing solutios for gathering data:

1. Compare to collecting data on cards or with a key-to-diskette system such as the 3741, the microcomputer can do a much more comprehensive job of editing the data. It not only can do checks on the type of the data but also on particular values of the data or by comparing values in several fields of the data and so forth. Furthermore, the microcomputer using a formatted CRT-type technology can much more readily be operated by users than just by the professional data entry operators. A recent survey by a professional data entry association shows that approximately 70% of corporate data processing departments are shifting to user data entry from centralized data entry. In terms of performance, the speed of key entry into a microcomputer usually exceeds, and sometimes by a considerable factor, the entry speed through cards and key-to-diskette systems (due to the fact that fairly smart duping operations can be programmed which can minimize the key strokes that need to be entered).
2. Compared to intelligent key-to-disk systems, the microcomputer would roughly equal them in terms of key entry performance since these machines also are intelligent and programmable and can provide for very smart data collection algorithms. The principal disadvantages of the key-to-disk systems, which are usually characterized by a minicomputer with a cluster of terminals, is that the per-terminal cost of the microcomputer is considerably less than the per-terminal cost of the clustered mini when you are dealing with fewer than 8 or 10 terminals in a specific location. For large scale data entry chores, one would probably still favor a clustered mini for doing the data entry chores. For operations with only a few stations or where the stations are distributed, the mi-

crocomputer again comes out as being a more favorable solution.

3. Compared to online data entry using V/3000 or some other online screen formatted technology, the microcomputer scores much higher in keystroke performance and operator performance as well as not burdening the machine with keystroke intensive work. I think it is the common experience of the HP3000 community that having several data entry operators entering data online disproportionately burdens that system's performance.

Again the issue of portability and versatility needs to be made in a comparison of data entry approaches. Virtually all other data entry gear is single purpose equipment. If you buy a key punch machine, or a 3741, or a key-to-disk system such as the Data 100 data entry system, you are buying specific hardware for a data entry chore oriented towards the data entry profession. A microcomputer data entry system carries with it all of the same benefits of an intelligent data entry system, but running on a general purpose piece of hardware that can be used for word processing, or running Visicalc, or other programs. Furthermore, the microcomputer-based data entry software, in general, tends to be more user-oriented and more oriented towards spreading applications out to the user rather than keeping the data entry function local to the data processing department.

HYBRID SYSTEMS

The most exciting systems on the horizon are those which are combining the online and offline capabilities which we have been describing; that is, systems that may operate sometimes online or offline depending on the desired properties of the system. Imagine an office of your company that has an APPLE or IBM Personal Computer or an HP125 sitting in it, perhaps performing word processing or financial modeling, but which can also be used as a data gathering station. What are some of the kinds of hybrid applications that we could make use of with this configuration?

Perhaps the most trivial example of a hybrid application is in performing a function we call error turnround handling. Imagine an order entry application that may have a variety of forms associated with it, perhaps representing an order header, repeating line items, and repeating partial shipments for each line item. Imagine that these transactions are gathered using the standard offline data entry software and are put into a batch, are checked and perhaps listed, perhaps batch totaled and are finally transmitted to the transaction processing HP3000. Clearly, there are going to be some database semantic errors which cannot be checked on the local microcomputers: credit limits might overflow, certain products need to be checked against the major database, and so forth.

The conventional solution for correcting these kinds of errors is to generate an error listing, send it back to the data entry operator, and have it rekeyed in the next batch. Our software provides an error turnaround mechanism by which the data that is sent to the transaction processor can have a tag put on each data segment that shows where it came from in the original source batch. When it detects an error in a record, the transaction processor can strip off this header fragment and write it to an errors file and gather up a file which represents all of the errors that were found. For example, out of 100 sales orders entered, three of them may have database failures which require further information to be entered. Using our communications software, this errors file can be sent back down to the microcomputer where a utility that we provide automatically runs that errors file against its original batch and creates an errors batch with only the three error records. Furthermore, the errors are now marked with the database codes such that the user can simply take them back into data entry system where a "Correct" command will automatically lead them to the field in error with an error message reported from the remote computer. This means that error turnaround information is not rekeyed but is simply sent back down and automatically creates a new errors-only batch. When this batch is corrected, it can then be sent up again to the transaction processor for reprocessing.

A more exotic second example which can accomplish the same function is as follows. Imagine the same data entry application except that at transmission the transactions are going into the transaction processor as they are being sent by the microcomputer. Thus, a record segment of an order is transmitted and immediately checked against the database in real time. As an error is detected, the transaction processor immediately brings the form back up on the remote computer and instructs the operator to repair it immediately. Notice very carefully that the very same form is being used in the online as well as the offline case to gather, correct, maintain, and modify the data. The operator learns only one interaction protocol whether operating in an offline or online environment. Using this mechanism errors can be corrected as soon as they are found.

The most interesting concept for hybrid systems involves an interactive transaction processor. Imagine a transaction processor which lives on the HP3000 and which can recognize transactions coming into it. Those transactions can either be in a batch or one at a time. In this type of a system, the software could be designed in such a way that depending on the preference or the needs or the requirements of the application a user may be offline or online. Again, consider our order entry example. An operator types in orders offline for an hour or two in the morning when suddenly a high priority order comes into the office. Using our hybrid system the operator can immediately put the microcomputer into an online mode, invoke the transaction processor and enter one order. The order is then entered immediately. The operator then drops offline, and continues entering more orders in the batch. Later, the operator

again makes contact with the remote computer and now invokes the very same transaction processor and dumps in the morning batch.

You can see from these exaples that there are an infinite variety of hybrid applications. The important thing to recognize is the fact that the difference between online and offline is a distinction that has been made out of historical necessity. Microcomputer-based systems suggest that the microcomputer is the real interface to the user's information network. How the microcomputer chooses to handle the transactions — that is, whether online or offline — merely becomes an applicatin dependency or a priority dependency. As time goes on, we will see that the importance of these hybrid systems will make for very user friendly systems for which the user need not get involved in many of the details that we now consider essential.

CONCLUSION

I remember as a child getting up early in the morning to watch physics programs on 1950-style educational television. At that time the concept of educational TV was that a television camera would be placed in a studio that looked like a classroom. A teacher, with a desk in front and a blackboard in back, with a pointer for the blackboard, would make all of the motions of a teacher teaching a class and the TV was merely an observer to the classroom. Compare that technology with the technology of a "Nova" or a "Cosmos" in which suddenly television is recognized as being a medium with its own powers of communication, with properties far different than the classroom teacher.

We have witnessed this very same phenomenon in the growth of microcomputer software. Microcomputer software has simply assumed the role of minicomputer and main frame software without our examining the special attributes of the new medium. Small computers add something dramatically new into the computer picture. It is important for us to recognize those things that they do well, not so that we an exclude certain types of software applications from a small computer, but to figure out how we can use the particularly strong capabilities of the microcomputer to work in a friendly, compatible environment with our existing communications networks.

Software Management Techniques

Janet Lind

There is currently much information available to document the fact that the cost of hardware is decreasing dramatically, but the cost of software continues to climb. When questioning the source of this problem, it is necessary to consider that many hardware functions are now being implemented in software or firmware. It is also true that computers are constantly being used in new applications, and computer users have increasingly sophisticated needs.

Today's software systems suffer from a variety of problems. Often they are delivered later than originally scheduled. The systems may cost more than the original projections. The software may not meet the user's requirements, or may be unreliable. When the need arises to correct or upgrade the system, the cost involved may be in excess of the cost of the original system.[18]

One of the most pressing problems in software project management is the lack of a well-developed structure for guiding the individual programmer. Instead of directing the programmer's activities, the manager can often only manage an idea until all parts of the project are completed. This problem arises from the fact that the only clearly defined point in the programmer's work is completion. More definition of the process is needed.[2]

There is no reason why software development should be exempt from the formats found in other engineering fields. Lab notebooks, design reviews, and failure and reliability analysis have proved their value.

The lack of a disciplined approach to software development may produce programs which are difficult to understand or maintain, affecting overall cost. Therefore it is important to develop a more rigorous framework to delineate the several steps in the programming process. Knowing the proper steps to follow will allow a programming team to develop more common objectives about the problem solution. This will improve the product and the group motivation by allowing the programmers to focus on more immediate goals.

Even though the approach being taken is to define a series of programming steps, it is always important to allow feedback to improve the product. A sequential description of program development steps will be defined here, but a problem found may cause a redefinition in preceeding steps to provide a more correct solution.[8]

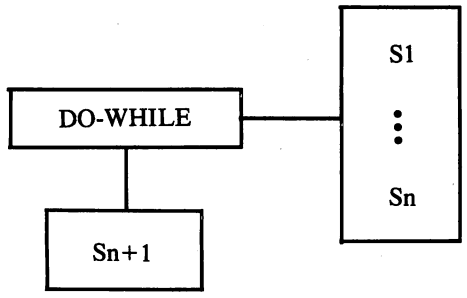
When first approaching a software project, it is necessary to perform a problem analysis. Here the inputs and outputs must be specified and the relationships between them must be described. A programming notebook should be kept to indicate how decisions were reached.[4]

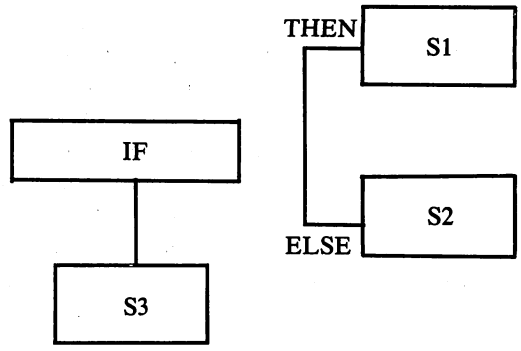
Part of the problem analysis includes decisions about the resources available. This includes both people and computer power. When considering the hardware used, it is no longer strictly correct to consider implementing everything possible in software. To increase productivity and to simplify code requirements, it may be worthwhile to purchase or develop hardware to meet the problem.

Another choice would be the use of multiple processors, which gives greater flexibility than implementing a function in hardware. This could also decrease the complexity of a given program, for it will no longer be responsible for as many portions of the function. Programs could also run concurrently, reducing timing constraints on a single system. This would make programming in higher level languages more attractive because the added processor capabilities offset the less efficient code produced. After completion of problem analysis, a walkthrough should be performed.

The solution design is driven by the I/O and their relationships defined in the problem analysis. Several different documentation techniques and evaluation criteria can be used in the structured design. Data flow diagrams can be used at the high level abstraction to model the flow of data through the system.[5]

Higher order software notation, or HOS, which was developed as part of the Apollo Program at Draper labs, defines a very useful flowcharting technique. Each control structure has a horizontal block showing the program flow in that structure. This type of flowchart does not show extra arrows, and allows easy identification of each possible branch. This notation also uses the same identation as should appear in the actual code.[3]





HOS Example

Some of the evaluation criteria used in structured design include decisions about the possible program development tools available. Certain programming languages may provide better support for the data structures to be used. They may also affect the amount of coupling required between modules. It is important to consider the capabilities of the computer system on which the program will be run, including memory management techniques and I/O capabilities.

When doing structured design, the design team is often tempted to perform just the top level abstraction as a team, designing the lower levels individually. There are some important reasons for doing a single integrated design of the entire application. First, subdivision of the design may result in excessive coupling of the major systems. The resulting packaging into programs from a subdivided design may be suboptimal. A complete overall structural design could produce more efficient and convenient packaging. Subdividing the design work will very often result in duplicate programming. It is particularly unfortunate when minor changes occur in a few structures, yielding a new system which could have shared entire subsystems and many levels of modules.[9]

Even though there are reasons for completing the entire structural design as a single unit, this is not always possible. In that case it would be best to produce a high level abstraction of the program flow and identify the more independent subsections. Those with few, uncomplicated interconnections could be treated independently. To avoid duplication of code, frequent mutual design walkthroughs and cross-checks should be performed.

Either while the structured design is being developed, or after its completion, the testing must be planned. It is necessary to design the test cases before the coding is begun. This allows peer review to verify that the designed code can be tested.

If the HOS flowchart notation is used, each program branch can be easily identified, and therefore tests can be designed to exercise each branch. If each program branch is numbered, a test matrix can be developed to indicate which tests execute which branches. The input and output to each test must also be specified.[4]

Both the structured code design and the test design should be carefully reviewed via structured walkthrough techniques. When considering walkthroughs, it is necessary to determine if it is more economical for an error to be found by the programmer, or by a group of 3 to 5 people. Part of the cost-benefit calculation is the turnaround time for repairing errors. Recent studies indicate that it is roughly ten times more expensive to fix a design error after it has been coded than to repair an error detected in design phase. It is also quite possible that when looking for errors, the programmer can repeat a logic error and never find the bug. Walkthroughs can help avoid this.[10]

Test Case	Input		Branch					Output		
	V1	V2	1	2	3	4	5	V1	V3	V4
1	0	0	X					0	0	0
2	0	1		X	X			0	0	2
	1	0		X		X		0	2	1
3	1	1	X		X		X	1	1	1

Test Matrix Example

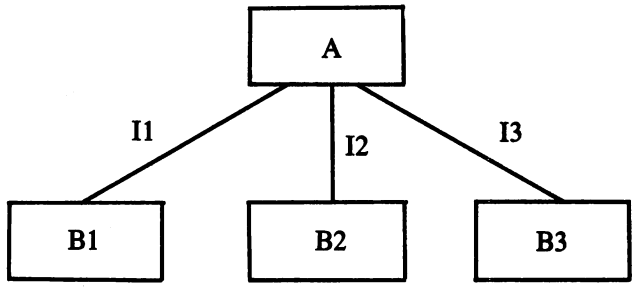
There are other walkthrough benefits which must be weighed against the cost. The product quality is improved. The walkthrough participants are better trained in the product and are able to exchange important information. This exchanged information increases the probability that the product can be salvaged if a programmer leaves before completion. A walkthrough is also a good environment for feedback into other areas.

After the designs have been accepted, coding and debugging can begin. Here structured programming techniques should be both understood and applied. Using the HOS flowchart technique makes program flow and structuring obvious at coding time.

It is too simple to believe that code without "GO-TO" commands is always good. The language being used should be well comprehended by the programmers to ensure that the proper constructs are used. The code within each module must be structured. Concurrent documentation should also be kept.

With developing and testing code, it is also necessary

to choose between a top-down or bottom-up approach to the overall structure. If hardware is being developed concurrent with software development, the lower level modules may be needed first to verify the hardware. In most other cases, a top-down approach can provide a more obvious visual presentation. This technique also allows modules to be tested together sooner. The interface between a node and its predecessor can be tested as soon as the lower level node is developed, allowing design or implementation errors to be detected and corrected earlier.[9]



Example

TOP-DOWN	BOTTOM-UP
1. Code and debug A	Code and debug B1
2. Code and debug B1	Code and debug B2
3. Test I1	Code and debug B3
4. Code and debug B2	Code and debug A
5. Test I2	Test I1
6. Code and debug B3	Test I2
7. Test I3	Test I3

A librarian function is helpful during coding and testing. The librarian can be an appropriately trained person, or an automated system. The librarian should maintain source programs and listings, as well as organizing all other technical information.

An automated system would avoid mixing media, which could be helpful in keeping a very accurate record of what changes are made. A record kept during edit phase could record what lines were modified and which variables were affected. A time stamp on this information could help other programmers know which version of code they were using. The knowledge that this system is being used will encourage a programmer to carefully analyze each change.

When the code can be tested, the test case matrix should be used to direct the tests applied. It may be useful to have the test run by the librarian. The test results should match those predicted, and a run log should be kept to document the test results. The purpose of the run should be stated, followed by an analysis of the run in terms of that purpose. This allows feedback for code correction and avoids haphazard modification. Any corrective actions which must be taken by the programmer should also be recorded.[5]

It may also be valuable to keep a time log to summarize the time needed for each step. This forces the programmer to review the actual effort expended in a task, and helps for making more realistic future estimates.

Throughout all activities, an independent auditing function can be performed. This will help detect errors unnoticed by the development team, and provides feedback.

The system described here is relatively involved and may be difficult to implement all at once. A pilot project could be chosen to use structured coding, structured design, and informal walkthroughs. As the process is implemented, it may be valuable to measure certain aspects such as the number of debugged lines of code produced per day and the number of bugs found after release. This can aid in future estimates. The amount of time spent in each walkthrough and the number of bugs found there should also be measured to help improve the techniques used.[6]

BIBLIOGRAPHY

[1]F. T. Baker, "Chief Programmer Team Management of Productin Programming," *IBM SYST. J.*, vol. 11, No. 1, 1972.

[2]F. T. Baker, "Structured Programming in a Production Environment," *IEEE Trans. Software Eng.*, pp. 241-252, June 1975.

[3]M. Hamilton and S. Zeldin, "Higher Order Software — A Methodology for Defining Software," *IEEE Trans. Software Eng.*, vol. se-2, pp. 9-32, Mar. 1976.

[4]P. Hsia and F. Petry, "A Framework for Discipline in Programming," *IEEE Trans. Software Eng.*, vol. se-6, no. 2, pp. 226-232, Mar. 1980.

[5]P. Hsia and F. Petry, "A Systematic Approach to Interactive Programming," *Computer*, pp. 27-34, June 1980.

[6]M. Page-Jones, *The Practical Guide to Structured Systems Design*, Yourdon Press, New York, N.Y., pp. 267-284, 1980.

[7]C. H. Reynolds, "What's Wrong with Computer Programming Management?," *On the Management of Computer Programming*, G. F. Weinwurm, Ed., Auerbach, Philadelphia, Pa., pp. 35-36, 1971.

[8]M. Walker, *Managing Software Reliability – the Paradigmatic Approach*, A. Salisbury, Ed., North Holland, New York, N.Y., pp. 32-41, 1981.

[9]E. Yourdon, *Managing the Structured Techniques*, Prentice-Hall, Inc., Englewood Cliffs, N.J., pp. 10-88, 1979.

[10]E. Yourdon, *Structured Walkthroughs*, Prentice-Hall, Inc., Englewood Cliffs, N.J., pp. 87-100, 1979.

Understanding Hewlett-Packard: A View From the Inside

Jan Stambaugh
Field Marketing Support Manager
Business Computer Group

This paper is a description of the presentation to be given at the Users Group meeting in San Antonio. It is not reflective of the actual information to be conveyed for two primary reasons. First, the presentation begins with a short, fun quiz. The answers to the questions in the quiz are revealed throughout the presentation. Since prizes will be given to the winner or winners, pre-publishing the answers did not seem to be advisable. Second, the information I wish to share with you is particularly volatile; it changes so frequently that I hesitate to submit three months in advance a paper which I know will be obsolete when you read it.

Several years ago I saw a movie entitled "The Universe" in which the camera began somewhere out in the universe, then focused in on the galaxy, the planet earth, the North American continent, a state and so on, all the way down to a molecule and then an atom. This movie is analagous to the way in which I will present my inside view of Hewlett-Packard.

I will begin by describing the company as a whole, its domestic and international operations, the distribution of its sales orders, the distribution of its sales dollar, and its corporate goals. From there I will describe the company's organizational structure, its six major product lines, its groups, divisions, and operations, and how one relates to the other.

I will talk extensively about the Business Computer Group, that part of the company which is responsible for the HP3000 hardware and software. I will review the customer interfaces to the company and tell you how you, as a user, can make yourself heard.

NOTE: Those who attend the presentation will receive copies of the overhead transparencies.

Structured Analysis

Gloria Weld
Hewlett Packard Corporation

In any programming project, there are three areas of partition: Analysis, Design, and Implementation. All three of these areas can benefit from a systematic, structured approach.

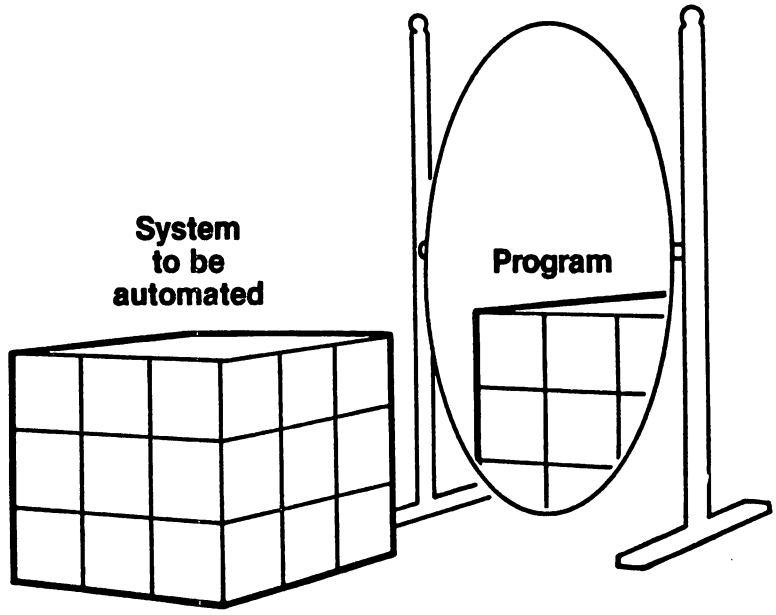
Today we will discuss Structured Analysis. In our discussion, our underlying assumption will be that we are called upon to design (automate) a new system in order to replace an existing system.

ALL OF US WHO ARE INVOLVED WITH PROGRAMMING AND PROGRAMMERS ARE CONCERNED WITH MAKING SURE THAT THE CODE WHICH IS WRITTEN ADEQUATELY AND APPROPRIATELY REPRESENTS THE SYSTEM WHICH IS TO BE AUTOMATED.

STRUCTURED ANALYSIS IS A METHOD TO ACHIEVE THAT GOAL.

Our Goal:

The program written must truly represent the system to be automated.



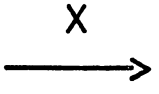
SOME TOOLS OF

STRUCTURED ANALYSIS

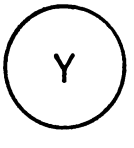
- DATA FLOW DIAGRAMS (DFD'S)
- DATA DICTIONARY
- STRUCTURED ENGLISH

DFD NOTATION

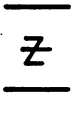
1. DATA FLOWS, REPRESENTED BY NAMED ARROWS



2. PROCESSES, REPESENTED BY NAMED CIRCLES I.E. ("BUBBLES")



3. FILES, REPRESENTED BY NAMED STRAIGHT LINES



4. DATA SOURCES AND SINKS, REPRESENTED BY NAMED BOXES

DFD'S

I. A LANGUAGE

II. AN EXCELLENT TECHNIQUE FOR UNCOVERING MISUNDERSTANDINGS DURING THE ANALYSIS PHASE OF A PROJECT.

COMMENTARY

HOW DO YOU ANALYZE A SYSTEM?

YOU TALK. YOU TALK TO THE PEOPLE

WHO ARE PART OF THE SYSTEM. YOU

ASK THEM WHAT IT IS THAT THEY DO.

Discussing "how things work" with a participant in a system can often lead to confusion. Quite naturally, there are multiple views of the system. Each participant in the system views the situation from his own vantage point. Thus, analysis derived from discussion with one participant will often conflict with analysis derived from discussion with another participant.

For example:

DESCRIPTION OF A "HOSPITAL SYSTEM"

A PATIENT COMES INTO THE HOSPITAL AND CHECKS IN. IF HE IS REALLY SICK, HE DOESN'T CHECK IN HIMSELF BUT HE'S PUT INTO A WHEELCHAIR AND SENT RIGHT UP TO A ROOM (UNLESS HE'S AN EMERGENCY-ROOM PATIENT). THEN THE DOCTOR ORDERS ALL THE LAB TESTS HE NEEDS.

SOMETIMES MATERNITY PATIENTS GO TO THE LABOR-DELIVERY PART OF THE HOSPITAL RIGHT AWAY.

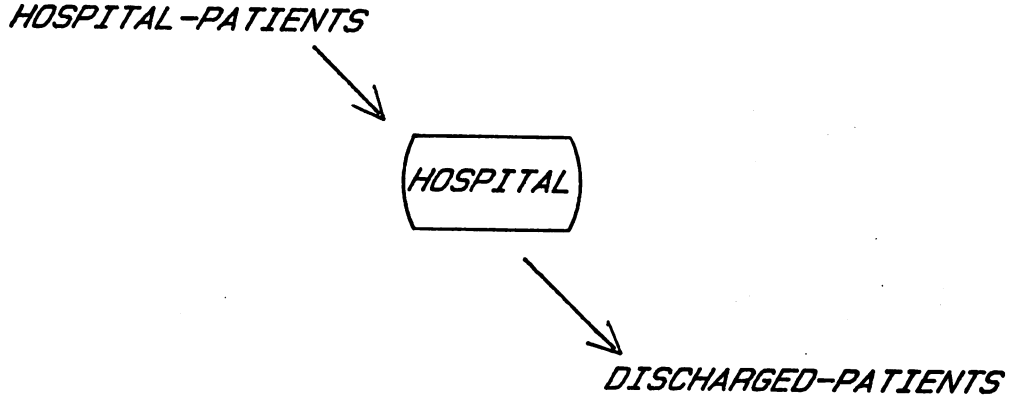
EMERGENCY ROOM PATIENTS HAVE TO WAIT IN THE EMERGENCY ROOM UNLESS THEY NEED TRAUMA CARE RIGHT AWAY.

AFTER TESTS AND X-RAYS ARE TAKEN, COPIES GO INTO THE CHART AND A COPY GOES TO MEDICAL RECORDS.

I AM CHIEF COOK IN THE HOSPITAL KITCHEN. ALL FOOD GOES THROUGH ME. EVERY DAY WE COOK BREAKFAST, LUNCH AND DINNER. WE COOK SPECIAL FOODS FOR PEOPLE WHO ARE ON SPECIAL DIETS, TOO. THAT'S THE HARDEST PART!

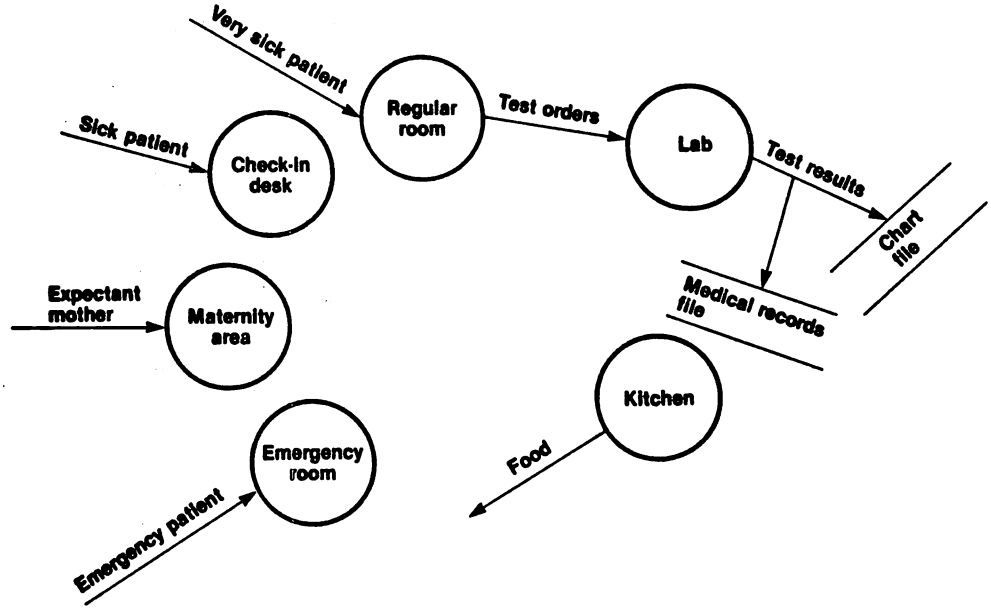
From this description, we can derive a "Top Level" of analysis:

HIGHEST LEVEL OF "HOSPITAL SYSTEM"



A "First Pass" DFD representing the HOSPITAL SYSTEM might look like this:

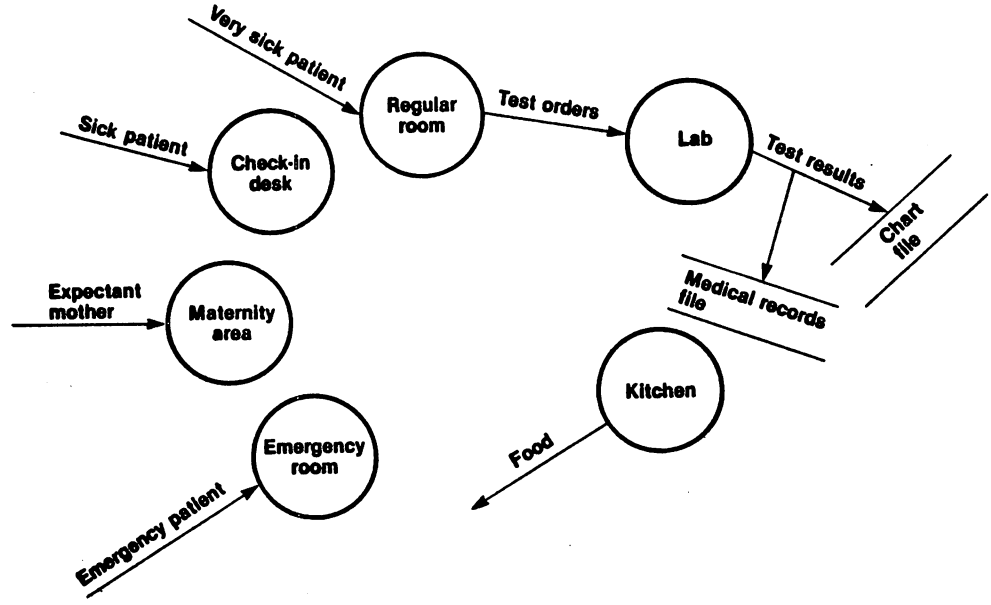
DFD of "Hospital System" Pass I (Taken from Verbal Report of a Participant)



As you can see, there are many empty spaces in our first pass DFD. From the description given us by our participant, we have created a DFD with data-flows entering process bubbles and no data exiting. We also have data flows coming out of process bubbles where no data ever entered.

Our "Tests for Correctness" which point out an incorrect DFD immediately point out to us that our understanding of this system is conceptually incorrect. And we (for the most part) know exactly what it is we don't understand.

DFD of "Hospital System" Pass I (Taken from Verbal Report of a Participant)

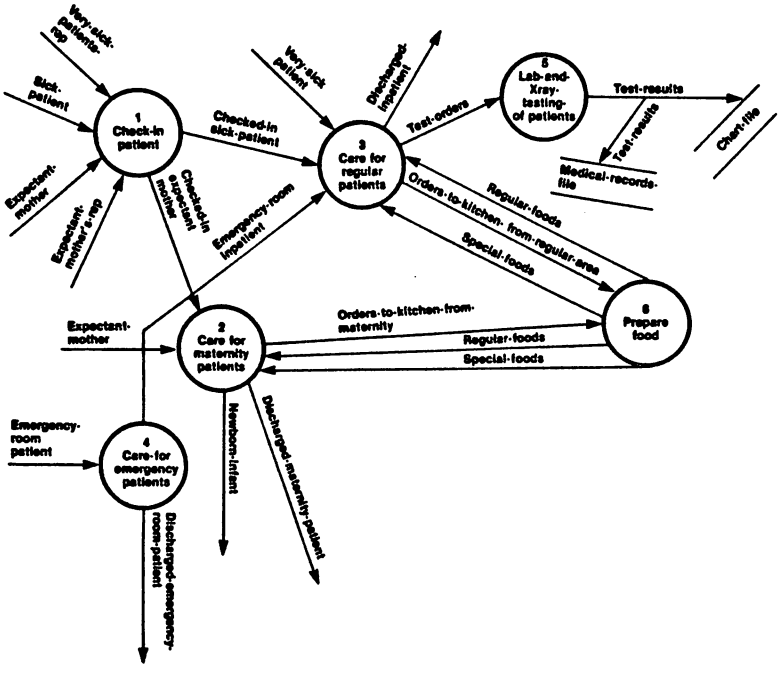


QUESTIONS WHICH COME UP WHEN TRYING TO ANALYZE THIS PASS 1 DFD

- WHAT HAPPENS TO A PATIENT WHO IS NOT VERY SICK? AFTER HE CHECKS IN, WHAT DOES HE DO?
- DOES A PATIENT WHO IS TOO SICK TO CHECK IN HIMSELF EVER GET CHECKED IN?
- DO EMERGENCY ROOM PATIENTS WHO DON'T NEED TRAUMA CARE EVER GET OUT OF THE EMERGENCY ROOM?
- HOW DOES A PATIENT (EITHER A REGULAR PATIENT, MATERNITY PATIENT, OR EMERGENCY ROOM PATIENT) EVER GET OUT OF THE HOSPITAL?
- HOW DOES THE KITCHEN KNOW WHAT SPECIAL FOODS ARE NEEDED? WHERE DOES THE FOOD GO ONCE IT LEAVES THE KITCHEN?

After asking those questions, we come to a DFD like this. True, it appears confusing. However, it is a pictorial representation of our system, a tool for discussion between the analyst and the participant.

DFD of "Hospital System" Pass II



We might wish to expand one of our process bubbles, in this case bubble number 5. (LAB AND XRAY TESTING OF PATIENTS).

Expansion of Bubble #5 in "Hospital System"

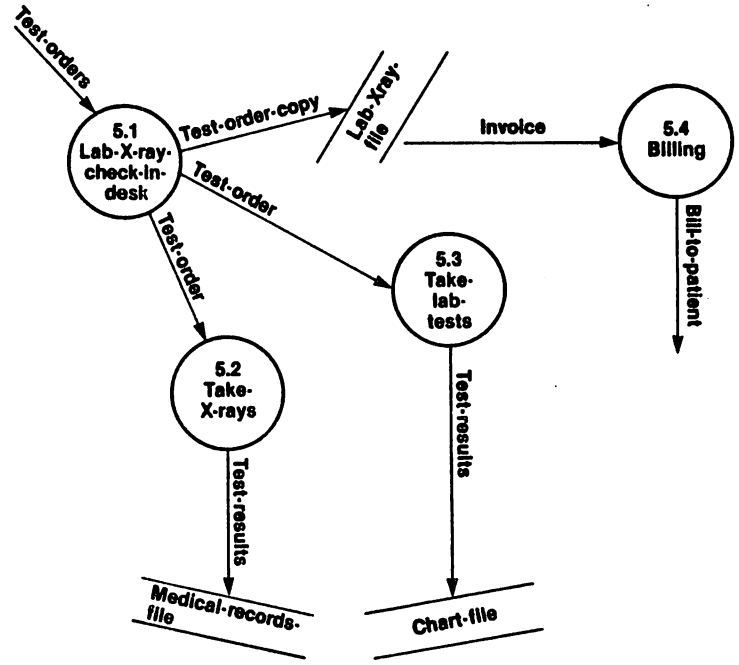


Diagram 5.0: Lab and X-Ray Testing of Data

Our "Test for Correctness" of this expanded DFD shows us that in the higher level DFD we had one input to process bubble #5 (TEST-ORDERS), and one output (TEST-RESULTS).

Here, however, we see two outputs! (TEST-RESULTS and BILL-TO-PATIENT).

Once again we immediately recognize an area of misunderstanding, and we return to talk to the participant in order to find out how the system really does work.

As we have seen, areas of misunderstanding can occur in data-flow path analysis. Also, there can be confusion about the exact definition of a particular data-flow file, or process bubble.

Structured Analysis contains a tool called the Data Dictionary, which attempts to eliminate ambiguity of definition.

DATA DICTIONARY

A SET OF DEFINITIONS FOR:

- DATA
- FILES
- PROCESS BUBBLES

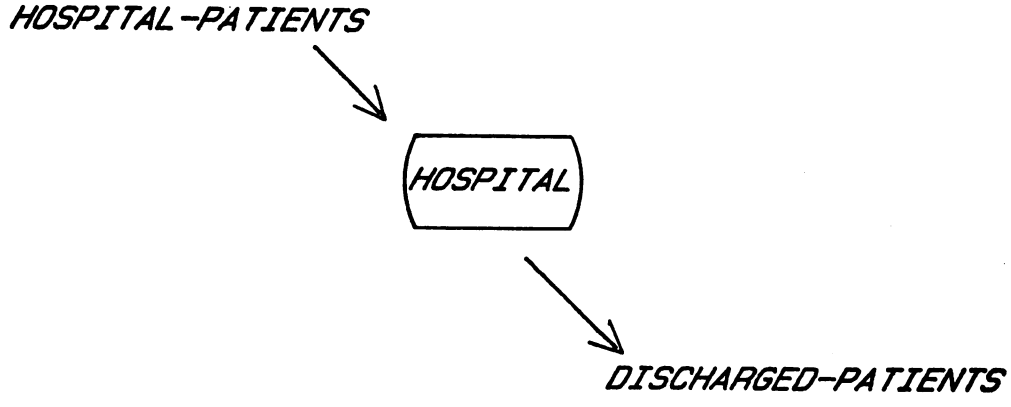
USED IN DFD

Here are some examples of Data-flow definitions in the Data Dictionary.

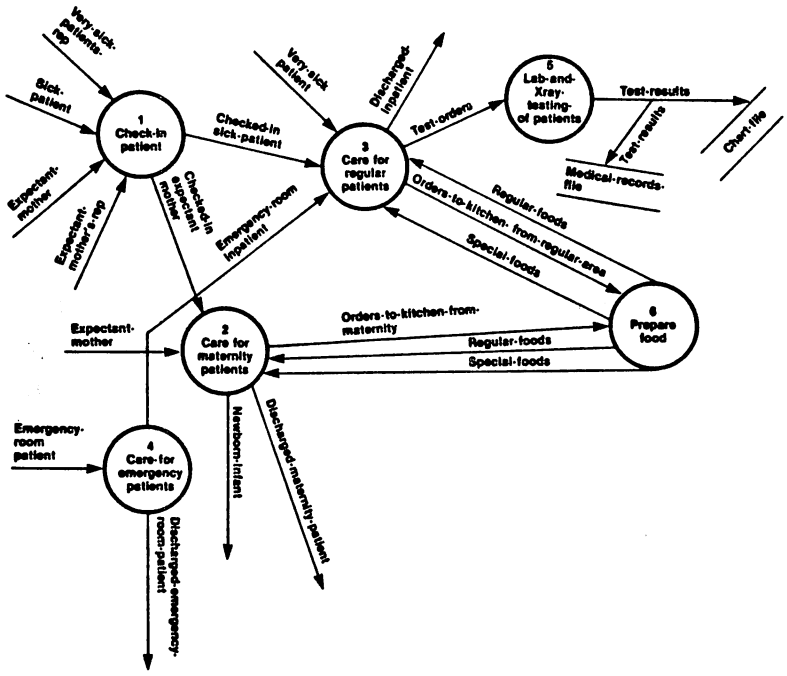
EXAMPLES OF DD ENTRIES FOR "HOSPITAL SYSTEM"

```
HOSPITAL-PATIENT          = SICK PATIENT OR
(COMPOUND OR GROUP)         EXPECTANT MOTHER OR
                            EMERGENY-ROOM PATIENT OR
                            VERY SICK PATIENT

DOCTORS ORDERS            = TEST ORDERS
(ALIAS)

EMERGENCY-ROOM-PATIENT    = "FLU"
(PRIMITIVE DATA             "AUTO-ACCIDENT"
 ELEMENT)                   "HEART-PROBLEM"
```

HIGHEST LEVEL OF "HOSPITAL SYSTEM"



DFD of "Hospital System" Pass II



Expansion of Bubble #5 in "Hospital System"

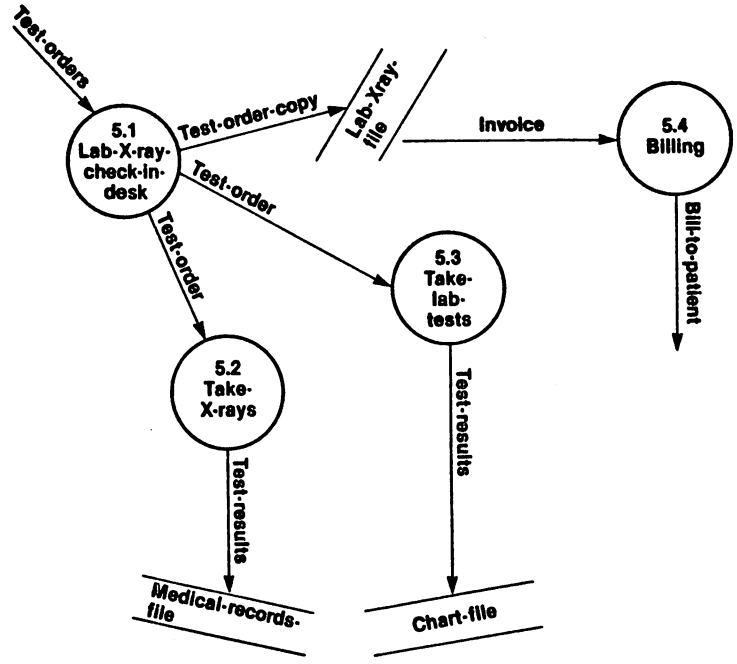
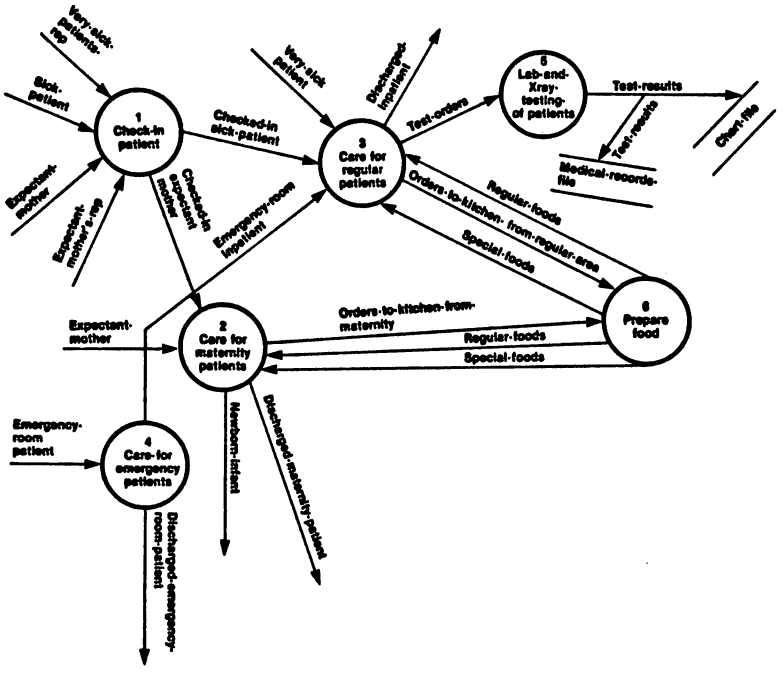


Diagram 5.0: Lab and X-Ray Testing of Data

DFD of "Hospital System" Pass II



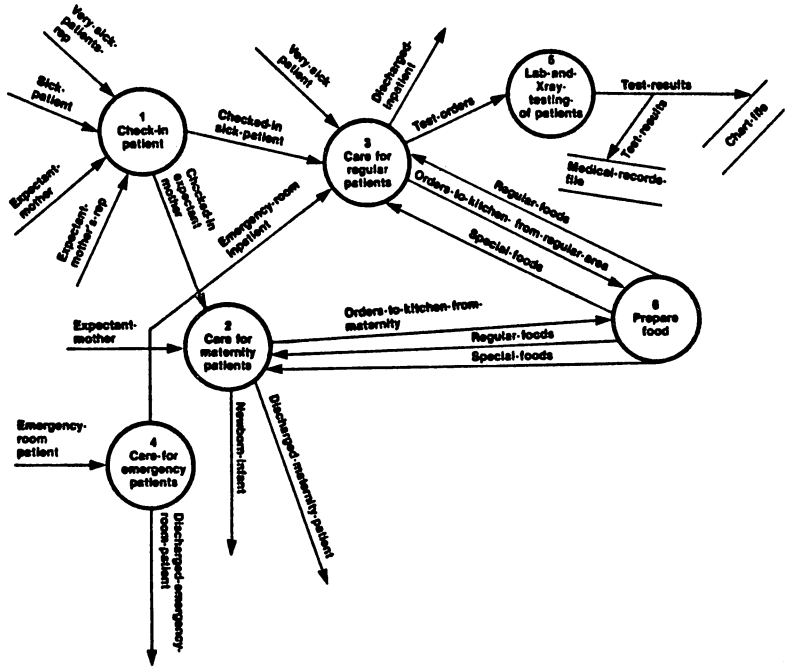
EXAMPLES OF DD ENTRIES FOR "HOSPITAL SYSTEM"

HOSPITAL-PATIENT = SICK PATIENT OR
(COMPOUND OR GROUP) EXPECTANT MOTHER OR
EMERGENY-ROOM PATIENT OR
VERY SICK PATIENT

DOCTORS ORDERS = TEST ORDERS
(ALIAS)

EMERGENCY-ROOM-PATIENT = "FLU"
(PRIMITIVE DATA "AUTO-ACCIDENT"
ELEMENT) "HEART-PROBLEM"

DFD of "Hospital System" Pass II



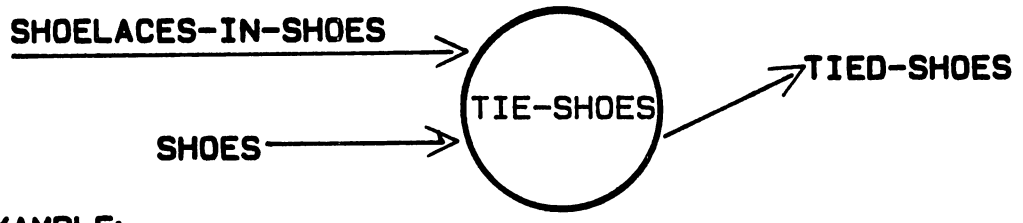
COMMENTARY

As we level our DFD for a system, each level of expansion shows more detail until we reach a level showing the primitive operations that act upon the data.

PRIMITIVE FUNCTIONS

PROCESS BUBBLES WHICH CAN NO LONGER BE EXPANDED REPRESENT PRIMITIVE FUNCTIONS WHICH ACT UPON THE DATA

EXAMPLE:



EXAMPLE:

Elements in the Data Dictionary which contain information about the process bubbles which are primitive functions are called Mini-Specs. Mini-Specs are written in Structured English.

Structured English

An Orthogonal Subset of English:

- Provides the minimum set of constructs needed to describe rules governing transformation of data flows for any functional primitive
- Provides one, and only one, possible way to describe rules governing transformation of data flows for any functional primitive

Policy for Preparing Foods

For each order-to-kitchen-from-regular area:

- For each special order:
  —Collect foods needed to fill order
  —Prepare foods
  —Send special foods back to appropriate room
- For each regular order:
  —Prepare foods
  —Send regular foods back to appropriate room.

In summary, structured specification consists of:

- DFDs — pictorially shows relationship within the system
- Data Dictionary — defines the data acted upon by the system
- Minispecs — describes the primitive function which make up the system. These are written in Structured English.

Our Data Dictionary is a rigorous description/definition of all Data Flows, files and primitive functions which occur in the DFD which was derived from our Structured Analysis of a system.

Structured Analysis is a large topic. In preparing this paper, the most difficult task was in deciding what information to leave out.

I would suggest if you have further interest in the topic of Structured Analysis and feel the technique could be of use to you that you consult the following references:

- *Structured Analysis and System Specification* by Tom De Marco, foreword by P. J. Plauger
- *The Practical Guide to Structured Systems Design* by Meilir-Page-Jones, foreword by Ed Yourdon.

An On-Line Interactive Shop Floor Control and Capacity Planning System

Walter J. Utz, Jr.
Lab Section Manager
Hewlett Packard

Production Management/3000, the newest HP Manufacturer's Productivity Network Application Product, was announced in the summer of 1981 with first installations in the fall of 1981. Production Management/3000 is an interactive application system for managing the production planning and control functions of a manufacturing operation.

This paper describes the major functions of the product, the contributions of the product, and the actual experiences at several test sites as the users integrated interactive shop floor control into their daily operations. The paper explains the uniqueness of the product, and the potential productivity gains which are inherent in the effective utilization of the product.

The challenges of managing production are very familiar to production managers (Slide 1). The efficient use of available resources to meet production requirements at the lowest total cost is a key to successful production management. HP's goal was to produce an interactive application system to manage production planning and control.

The process of improving manufacturing productivity through the effective management of production resources can be viewed as a cycle of causes and effects as illustrated (Slide 2). In the past, one of the major obstacles to successful management was getting timely information on inventories, work-in-process, and capacity requirements.

The six functional modules of Production Management/3000 are routings and workcenters, work order scheduling, work order tracking, shop floor dispatching, work-in-process, and capacity requirements planning. The environment best suited for Production Management/3000 is a manufacturer with workorders for fixed quantities of specific products with individual start or completion dates (Slide 3).

Data collection on the shop floor is available via the standard terminals, or through the factory data capture terminals (Slide 4). These special terminals are suited to users who are unfamiliar with computers or typing; they utilize a set of pre-defined functions keys and prompting lights to assist users with each transaction. The factory data capture terminals can be equipped with bar-code reading wands, badges, punched cards, or magnetic card readers in order to streamline certain types of data entry on the shop floor.

Production Management/3000 also features a customizable user interface, with features similar to those offered in Materials Management/3000. (Slide 5). The user is able to customize data, screens, reports, security, system values, "help messages," and processing specifications, without programming. Customization offers the user increased flexibility, shorter implementation time, and a lower maintenance burden.

These features highlight a brief summary of the functional capability of Production Management/3000. Let us now examine the ways in which this application tool can be used on the shop floor.

The physical appearances and layouts of discrete manufacturing shop floors varies widely, but the functional requirements are similar. The discrete environment is characterized by the workcenter; a typical example is shown in this picture (Slide 6). In this environment, the work is dispatched to a workcenter, the specified tasks are performed, and the work moves to the next assigned workcenter. Production Management/3000 has sophisticated scheduling capabilities, but they are not discussed in this paper. The point to be noted here is that the workcenter manager can review the work-in-process, priorities, status, etc. The workcenter manager can also track the work completed, rework, scrap, exceptions, labor data, and routing lists. In other words, the manager has an on-line system which eliminates the need for most written reports, although printed reports are available, if desired. Our test sites have now learned to check the terminal in order to resolve all questions regarding work-in-process, as well as scheduled work. They have also learned to quickly scan status information to look for potential trouble spots.

In a typical installation, every workcenter ould have a CRT, plus one or more factory data collection units. It should be noted that the system can be run entirely with CRTs, but the factory data collection units offer specialized features which make them attractive in many situations. Work order tracking would typically involves one or more data collection stations at each workcenter (Slide 7); a workcenter is composed of one or more workstations. The idea here is to allow the

employees to record work completion, labor, and status information. Our experience has been that one data collection station typically can support 15-25 employees, although there are exceptions where factors such as distance must be considered. The key point is to place the data collection stations where they are easy to reach. In those instances where employees have expressed discontent, it has almost always been resolved by moving a station closer to the employee, or adding another station.

Employee training is the key to acceptance of the system and the data collection concept. The best technique for training the shop floor employees is to conduct the training in small groups of 3 or 4. This gives each person a chance to ask questions, to try the terminal or data collection unit, and to understand what is really happening. Large training labs usually result in one or two self appointed leaders doing all of the terminal operation, while the others stand back and watch. Our original estimates of training were not enough; we have worked with our test sites in developing training to a point where it is most effective.

The workcenter managers pose a slightly different training problem. Many of them have been doing very good work for years, and they are somewhat resistant to change. Why fix it if it works? The managers are often quick to change their position as they come to realize that this on-line tool can really improve their productivity. In many cases they become ardent converts who then sing the praises of the system.

Here is where the realization of the uniqueness of this application becomes apparent. The manager can now see exactly what happened during the shift that just ended. If action is required, it can be taken in time to be effective. If a problem is developing today which will affect final assembly areas next week, the managers know about it now. The manager can also use capacity planning to determine potential trouble spots ahead of time. He can know that his workcenter faces a demand that is 150% of capacity four weeks from now. In the past the manager was not able to see this potential overcapacity problem. The first time the capacity report is available, the capacity problem may come as a shock. The manager's initial reaction is not to know what to do; however, the manager soon learns to modify his managerial skills to react to situations now, rather than being forced to wait until the problem has reached epidemic proportions. When the system is first installed, the managers have mixed reactions. As the system has had six to twelve weeks of operation, they become highly favorable. At the end of this period, they begin to wonder how they managed to operate the old way (Slide 8). The ability to know what is going on at all times and modify the course of events before things go wrong allows a degree of fine tuning which appeals to the skilled manager.

This degree of success requires that everyone uses the system. Our test sites have reported up to 98% accuracy in all shop floor transactions. This very high degree of accuracy allows management to use the system with the highest degree of confidence. The system is currently running in environments ranging from single shift to three shifts in production around the clock.

One of the best ways to improve productivity on the shop floor is through the use of capacity requirements planning. (Slide 9) The bottom line of the manufacturing productivity equation is the full utilization of assets without the need for idle equipment and inventory. Capacity planning permits the evaluation of the production plan before material and other resources are committed to production. In this way capacity planning allows you to smooth the production plan. The ability to track production on the shop floor allows management to ensure that day to day operations are following the capacity plan.

Prior to this time one of the ways in which to ensure a constant flow of product seemed to be to "flood" workorders into the front end of the system, in hopes that sufficient production would result. The excess of workorders could result in complications which were worse than the cure which was intended. (Slide 10) Production Management/3000 uses both open work orders and suggested work orders (from Materials Management/3000, or some other system) in the capacity plan. This results in a projection of workload over time for each workstation. An analysis of this load profile allows management to focus attention on potential trouble spots.

As we mentioned earlier, our experience with test sites has been that capacity planning has opened a whole new vista of planning. The first reaction is to assume that perhaps the application is simply furnishing data which outlines some potential trouble that might never occur. But the users quickly come to believe that the predicted will indeed become reality, if they do not take action. They have also come to understand that monitoring of day to day operations, and comparing them to the plan, is essential in order to keep production flowing smoothly.

Production Management/3000 can be run with Materials Management/3000, or it can be run standalone with orders entered from an external system. (Slide 11) Our test sites are running the product in both modes. The only difference that we have noted here is that those users who already have Materials Management find it quite easy to implement Production Management as they are used to the screen formats, customization, monitor, and the other features of the system. In any event, the training requirements for the shop floor personnel remain the same, as Materials Management does not use factory data collection units on the shop floor.

We have briefly discussed customization earlier in this paper. Customization is a major technological contribution which cannot be given the recognition it deserves in this paper. However, it should be noted that our approach in Production Management/3000 was to

put all of the product features in, and let the users remove those features which they did not require (Slide 12). This is evident in the screens as all of the features of the product are addressed. Out test sites have been pleased with this approach, and they find it easier to tailor existing screens to their needs, than to try and put in numerous additional functions. The editing has been done by the users running the customizer, and the process does not require computer specialists.

The same situation has been experienced in operations, where persons with a minimum of computer experience have been able to operate the system. In fact, one of the test site operators has become skilled in using Query to generate additional reports as needed.

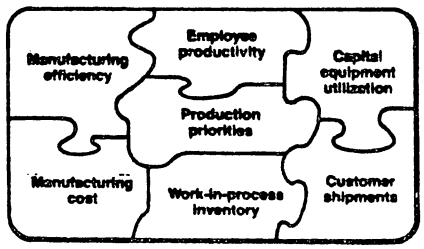
In general, the test sites were pleased to discover that their own non-EDP type manufacturing personnel could learn to use the system quite easily.

When Production Management/3000 was in the design stages, we knew that we were developing a major new product which should make a significant contribution. What we did not realize was the discrepancy between the environment of many shop floors and the solution which we were inventing. Many of the existing systems are batch oriented with reports appearing long after there is any time to do anything about it. In fact, workcenter managers have developed their own survival techniques for this environment which include banking some of the punched cards during good weeks for submission at a later date during a bad week. The concept of having all production information on-line, and updated at all times, comes as a bit of a shock. (Slide 13) The managers quickly discover that the new environment works to their advantage, but this conversion is complete only after the system has been in place for a month or two.

Another area of surprise comes when the routings and workcenters are being entered into the system. At this point the managers take a look at their operation and discover that in some cases it is far from efficient. One example is where the transit time between two workcenters greatly exceeds the actual time in the workcenter. The managers are tempted to change their shop floor based upon these initial insights into potential inefficiencies. Our advice to the users as been to go ahead and implement the application based upon the present shop floor, rather than try to correct things now. After all, production as been working, and too many changes all at once could complicate matters beyond repair. Better to get the application up and running, make a detailed study of all of operation based upon data gathered by the system, and then make improvements.

Production Management/3000 can be used to solve a variety of problems. The potential offered by customization gives great flexibility to the product. The localization of the product into many foreign languages expands the potential even further. Our initial experiences with test sites have indicated great satisfaction with the functionality of the product, and a wide variety of benefits which have resulted from its implementation. As the number of installations grows, the users will develop the expertise which will enable them to gain the greatest benefits. (Slide 14) We have seen the awareness of the potential of this new application begin to emerge, and we are anxious to work with the users to bring the product to its full potential.

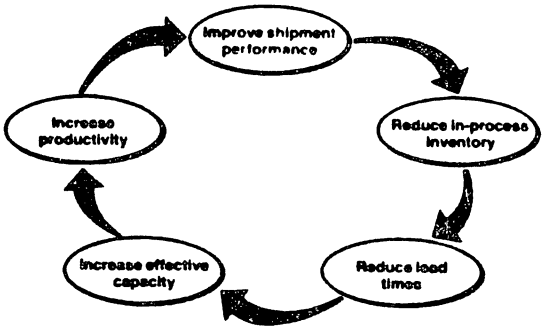
The Challenges of Managing Production



Balance Production Issues to Increase Manufacturing Productivity

Slide 1

Production Planning and Control Helps



Slide 2

Production Management/3000

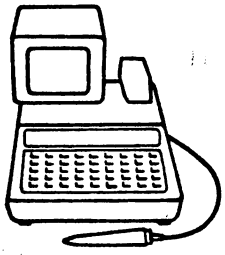
Schedules, tracks, and plans capacity using

workorders
for
fixed quantities
of
specific products
with
individual start or completion dates

Slide 3

Factory Data Capture Terminals

- Easy to use
- Desktop or wall mounted
- Multi-media input
- Alphabetic keyboard layout
- Prompting lights
- Function keys



Integrated Factory Data Collection

Slide 4

Customizable User Interface

Modify HP Manufacturing Systems

- Customize data base
- Customize data entry and retrieval screens
- Customize reports
- Customize data and device security
- Customize system values
- Customize "Help" messages
- Customize processing specifications

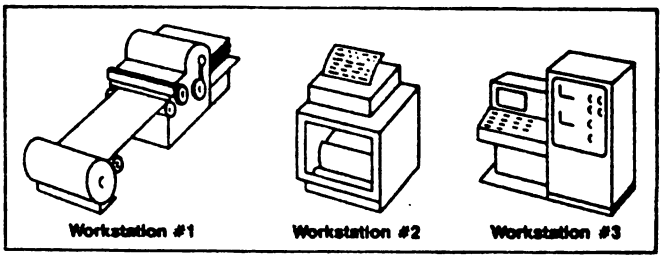
. . . without programming

Slide 5

Routings and Workcenters

Flexible two level facility definition

WORKCENTER

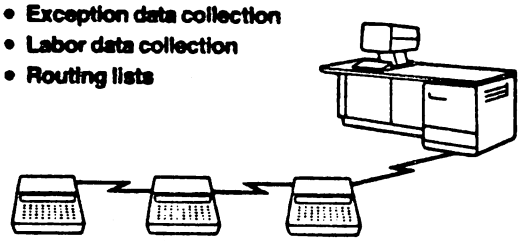


Match your equipment layout AND your organization

Slide 6

Work Order Tracking

- On-line sequence completion reporting
- Rework and scrap data collection
- Exception data collection
- Labor data collection
- Routing lists

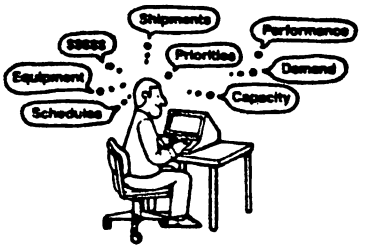


Accurate shop status is available at all times

Slide 7

Work-In-Process Control

- Evaluate manufacturing performance
- Analyze shop status • Fine tune the production plan

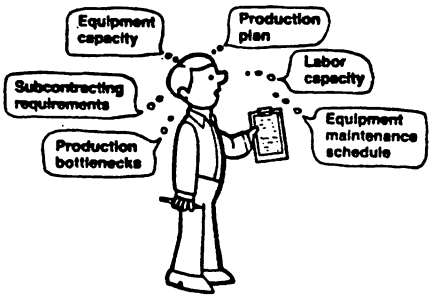


Balance Production Issues to Increase Manufacturing Productivity

Slide 8

Capacity Requirements Planning

Helps you evaluate the production plan

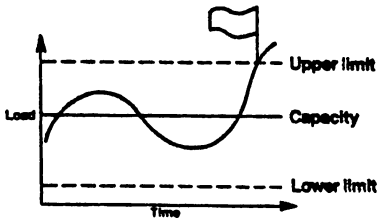


Realistic Production Schedules

Slide 9

Capacity Requirements Planning

- Workstation load profiles
- Exception reporting
- Alternate load measures
- Workcenter labor summary

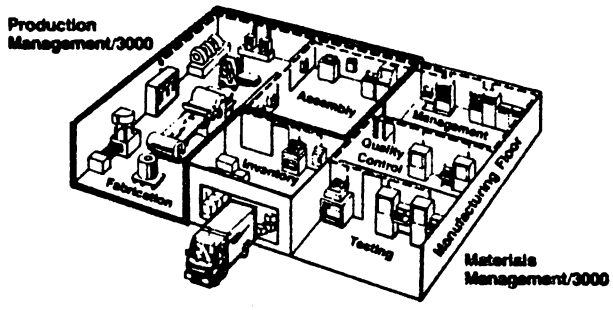


Predict Labor and Equipment Requirements

Slide 10

Production Management/3000

Completes the Manufacturing Planning & Control Cycle

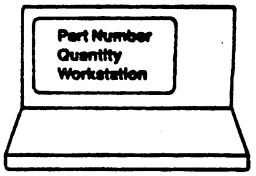


Slide 11

Customize Screens

Users can . . .

- Change existing screens
- Design new screens
- Change sequence of screens

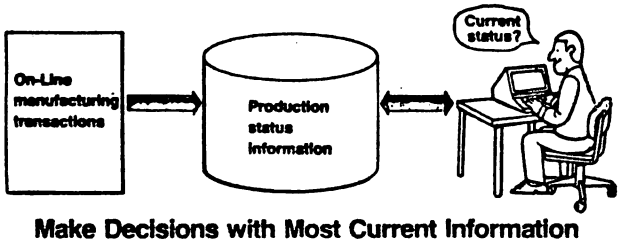


...with HP Manufacturing Systems

Slide 12

Manufacturing Information is Timely and Accessible

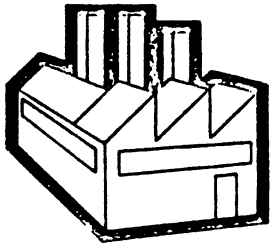
- On-line entry
- On-line data base update
- On-line information retrieval



Make Decisions with Most Current Information

Slide 13

HP Manufacturing Systems



Interactive Application Products to Increase the Productivity of Manufacturers

Slide 14