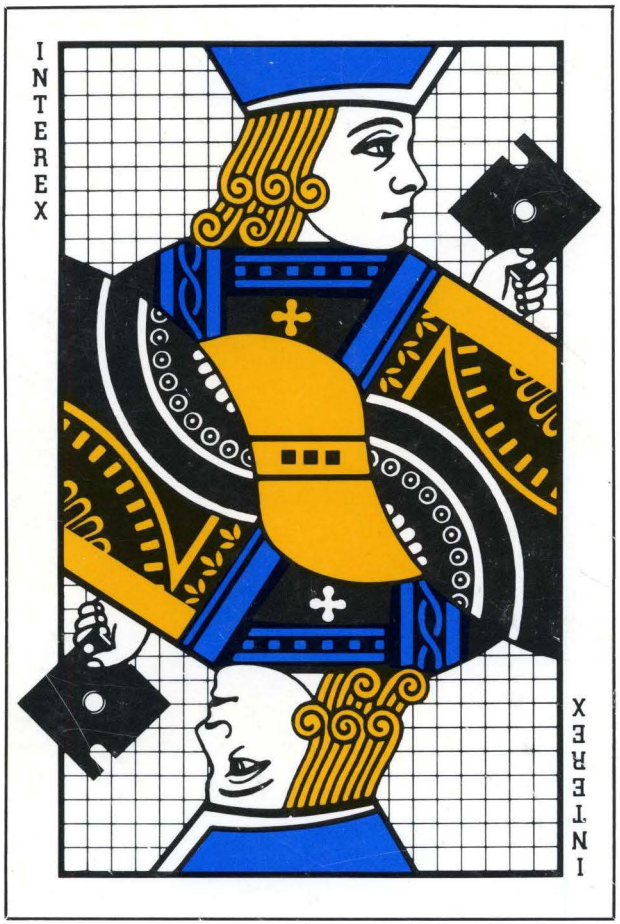
INTEREX



HP BUSINESS USERS
CONFERENCE PROCEEDINGS
VOLUME 2

LAS VEGAS

SEPTEMBER 20-25, 1987

INTEREX

the International Association of
Hewlett-Packard Computer Users

Proceedings

of the

1987 North American Conference
Of Hewlett-Packard Business
Computer Users

at

Las Vegas, Nevada
September 20-25, 1987

VOLUME 2

What If ... You Didn't Wait For Spectrum?

-or-

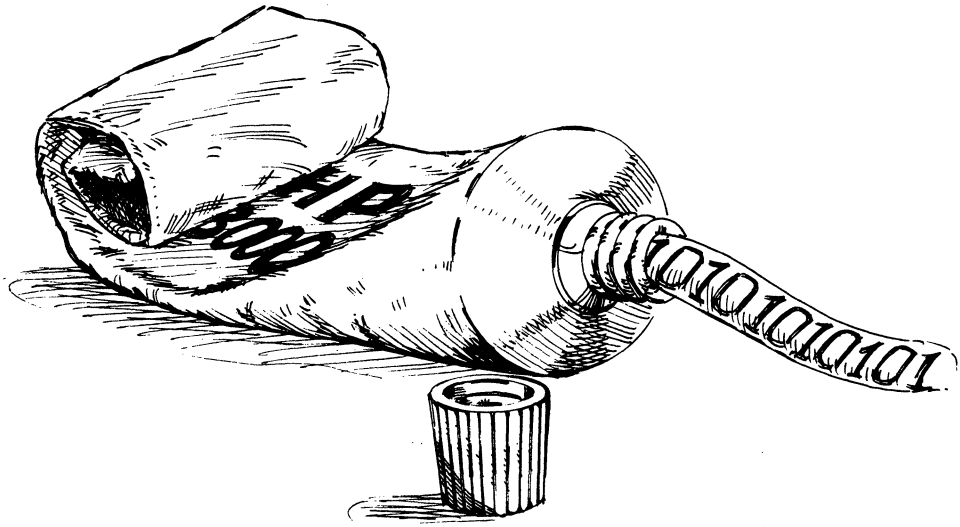
Squeezing the Last Bit Out Of Your HP 3000

By Michael Shumko, Robert Green, and David Greer

Robelle Consulting Ltd.
8648 Armstrong Rd. R.R. No. 6
Langley, B.C. Canada V3A 4P9
Telephone: (604) 888-3666
Telex: 04-352848

Copyright Robelle Consulting Ltd. 1987.

Permission is granted to reprint this document (but not for profit), provided that copyright notice is given.



What If ... You Didn't Wait For Spectrum?

-or-

Squeezing the Last Bit Out Of Your HP 3000

By Michael Shumko, Robert Green, and David Greer

The problem. Nightly batch jobs are still running the next morning. Users are complaining that on-line response is terrible. In short, your HP 3000 is over-worked, underpaid, and about to collapse from exhaustion.

The solution. Order a Spectrum: a Series 930 or 950.

The problem with this solution. Neither machine exists yet. (Okay. In the lab. But if it's not on the showroom floor, it can't be bought.) What to do?

The solution to the solution. We have done an informal survey of large HP shops to find out how the successful ones avoid topping out the HP 3000 line. What we found was not some "secret" formula, but rather a mundane, continuous, attention to the details of system performance. The successful sites still apply the long-touted answers for boosting performance, such as balancing use of disc drives. Just look in your magazines, newspapers and conference proceedings for all sorts of ways to improve performance.

Here are some of the ideas these users mentioned for how to 'squeeze the last bit from your HP 3000'.

1. pay attention to the details.
2. use one cpu per problem (distributed processing).
3. distribute an integrated solution over several CPUs.
4. put heavy cpu work on PCs (word processing, graphics).
5. upgrade to faster hardware (Series 70, LAN, forms cache).
6. review batch processing.
7. use NOBUF tools and optimum block sizes.
8. compile your fourth-generation applications.
9. get OMNIDEX for fast on-line database searching.

Not all of these solutions will apply to everyone. Many of these ideas are "old hat", but they work. A few of these ideas are novel - you may not have heard of them before. Some are not cheap (then again, neither is a Spectrum). If you're strapped for horsepower now, then these timely suggestions may give you the breathing room you need. Until Spectrum, of course.

Good system managers never stop thinking of new ideas to improve system performance. Successful sites are constantly monitoring their machines. Here are some of the tools they use.

Response Time.

Most shops we surveyed had OPT, but most were not using it regularly. Users complain that OPT is difficult to understand (even with all of the training). Joe Ballman at Textron Marine Systems thinks that OPT consumes so much CPU time that it affects its own measurements. David Lustig of BOSE uses a simple method to measure response time. When a user complains that the computer is slow, he goes to the users terminal and uses a stop watch to time the actual response time. Other sites were using SUE or SURVEYOR from the contributed library.

HPTREND.

Most sites are using HPTREND to provide accurate information about machine usage. In many cases, the HPTREND reports confirm the system manager's intuition and provides concrete evidence for upper management. Sites we contacted are planning their machine resources at least a year into the future. Jim Bird at Turbo Resources is trying SYSPLAN from Carolian Systems. This product is similar to HPTREND, but the trend analysis is done on your own machine.

Database Performance.

Turbo Resources uses HowMessy to indentify the inefficient datasets in their application (HowMessy is run once a week). Turbo uses DETPACK from Adager to repack one critical dataset every day. HowMessy was used to obtain the "before" and "after" pictures of the dataset packing.

Disc Cache Optimizer.

Markku Suni, an SE in Finland, has written an unsupported program which manages disc caching parameters dynamically. It varies the sequential and random fetch quantums depending on the current job mix, I/O queue lengths, etc. It will even disable caching on a drive if it decides that throughput would increase without it. DCO does not work well when the machine load is extremely dynamic (e.g., on a development machine). You can obtain a copy from your SE.

These are a small sample of the ideas that help monitor and improve system performance. No one knows all of the details that will keep your *machine running; you must strive to find them.*

Use One CPU Per Problem — Tip #2

Problem.

How do you add CPU power to a 3000 when you already have a Series 70?

Solution.

Use one CPU per problem, or application, or department. Don't try to crowd everything onto one computer. Instead, use a separate CPU for each major application, or give each department its own machine. That way, you make each application independent of the problems in other applications. If the Payroll application is a hog, there's no reason for the Accounting users to suffer. Using separate machines also allows you to tune each machine for its own application. 'Distributed processing' was the strategy most frequently mentioned in our survey of successful sites. Most give programmers their own machine.

Examples.

At Boeing, one of the large manufacturing systems has an 'update' machine and an 'inquiry' machine. The 'update' machine has 150 users who are updating the database. No uncontrolled inquiries or reports are allowed on this CPU. The 'inquiry' CPU has a copy of last night's database from the 'update' CPU; on this machine they allow people to make inquiries and to run QUIZ.

HealthPlus of Michigan provides heath care services using a Series 70 with 52 sessions for all data entry and a Series 68 with 30 sessions for all on-line inquiries and reports. They use Silhouette to keep the inquiry database current and a Series 48 is reserved for all program development. Word processing is done on two stand-alone Series 37 machines.

Longs Drugs, a large west-coast chain of drug stores, has 200 HP 3000s. An extreme example? Not really. True to the distributed processing ideal, each store has its own Series 37. These handle the main pharmacy application, keeping track of prescription stock, filling orders, and checking for dangerous drug interactions. When required, the Series 37s use dial-up DS to exchange information with the head office Series 70s. Otherwise, they're stand-alone machines. Every machine has a Console Engine to let the head office know when problems occur. (In fact, the Console Engine was initially developed for Longs Drugs.) At the head office, Longs puts separate applications on separate machines. For instance, all the Personnel applications are on one Series 70, the Accounting applications on another. Development is done on a separate machine again.

Consider another example, a company that sells supplies. They have 18 HP 3000s spread all over the world. Before the MIS manager went to work there, his philosophy was always 'get a bigger machine'. Then he went there, and they have a philosophy of 'getting the data down to the users'. So they have 3000s everywhere; every warehouse has its own small HP 3000. They were having a problem with FA/3000: they gave it its own Series 58. They don't even have a Series 70, and aren't budgeting for one until fiscal '88.

Tools.

If you go this route you'll want to make sure that you have the proper tools to manage the network of machines properly. One type of tool is used to route spool files from one machine to another conveniently.

Unispool from Holland House is one example of this. This allows you to have an expensive peripheral like a laser printer connected to one machine, and have more than one computer send output to it.

RSPOOL, from the contributed library, will duplicate spoolfles across a DS line. RSPOOL creates a remote session, runs a remote program to generate the remote spoolfle, and purges the original spoolfle. The price is cheap, but Joe Ballman of Textron Marine warns that RSPOOL eats up the LAN and consumes over 3% of the cpu.

$Stdlist Management software (now called Job Rescue) from NSD can also help; it checks spool files for error messages. This lets the computer look for problems itself, allowing the users to get on with their own work instead of baby-sitting the computer.

If you setup a machine in a user department for unattended operation, you'll still have console messages to contend with. The Console Engine from Telamon attaches to the console and looks for specific conditions such as system failure messages, error conditions, and that sort of thing. If it sees that the system has run into some trouble it can either take action on its own (a 'pseudo operator') or it can dial the head office and notify the system manager.

Resist getting a bigger machine.

You can always have that in reserve if you get in trouble. Get another machine instead. Dexter Shoes manages one million open items, one million inventory items, six shoe factories, over 50 retail outlets, and numerous wholesale clients with a network of six Series 40s. 'The key advantage that system managers see to the "one cpu per problem" philosophy, in addition to never 'topping out', is that you can push the machines into the user environment. You don't have to have a giant MIS. And when the machine is slow, it's because the users are running QUIZ reports. There are only a dozen users, so they can observe and figure it out, whereas on a Series 70 with 180 users, even the system manager doesn't know what's causing the problem. So you break it into smaller problems. Each machine is less complicated, and we would guess, has fewer problems. You will pay a little more for maintenance and raw horsepower, but you should be easily repaid in better user service.

Distribute An Integrated Solution Over Several CPUs Tip #3

Okay, I accept the idea that I should have one application per CPU, but my application is an integrated solution. All of the modules access common databases and I don't have time to rewrite it (or I bought the package and I don't have the source code).

Problem.

You can't split a single integrated application over two machines.

Solution.

Yes you can, if you are clever.

AutoNet.

Karl Smith of Softsmith has developed an ingenious, simple method of distributing an integrated application over several HP 3000s. Compaq Computers started in business a few years ago. To manage their manufacturing work, they used ASK's MANMAN system over dialup lines to a time-shared Series 42. Within weeks they had their own machine, then two, and so on. Their growth has been so dramatic that they have never had the time to customize ASK's programs -- they use them "as is". Compaq now runs its entire company on a network of 900 PCs and seven HP 3000s (no IBM mainframe). When their processing needs for MANMAN exceeded the power of a single Series 70, Tom Callaghan hired Karl to program a solution.

Tom wanted to be able to spread the databases and files of the integrated MANMAN application over more than one HP 3000. Karl wrote an SL routine to intercept all calls to the FOPEN intrinsic. His routine, called Global FOPEN, checks the user's desired filename against a table of remote file-set names. If it doesn't find a match, Global FOPEN calls the real FOPEN. If it does find a table entry for the filename, Global FOPEN automatically gets the user a remote session with the same logon as his local session (unless he already has one), and calls FOPEN for the remote file. With this method, Compaq can easily distribute the ASK MANMAN package across several machines, with *no changes to the application.* Karl advises that there be a logical split in the application, where files may be moved. In the case of MANMAN, the three major components are purchasing, manufacturing and physical inventory. Users logon to the machine which contains the component they are interested in. This ensures that most of the database access is local, with only occasional access to files on the other systems. For more details, contact Karl at (713) 332-3846 and ask about "AutoNet".

The Inside Details.

The software is not terribly tricky after all. The normal FOPEN is renamed to be HP'FOPEN, and the Softsmith FOPEN routine is added to the system SL. When FOPEN is called, this routine determines which system the requested file resides on. If it's on another system, it just inserts the DS machine name into the device parameter, then calls HP'FOPEN. Nothing to it. If necessary, it opens a DSLINE and does a remote hello onto the other machine. In UB-Delta-1 the remote logon can be done automatically by NS as part of the DSLINE command, making Karl's routine even more vanilla. There will still be a remote CI process. All that is saved is the trouble of having to do the

remote hello and remote bye. Another advantage is that this new feature takes one less NS socket.

Reflecting Mirror Images.

Miles Gilbert was designing a new Accounts Receivable system in Transact for Dexter Shoes. Unfortunately, the people responsible for names and addresses were in Boston and the people responsible for transactions were in Maine. How could Miles put the data near the responsible users when both groups needed access to all of the data?

First, Miles split the database in two: names/address versus transaction. Then he put a Series 40 in each location, with both databases on both machines. The users in Boston maintain the name/address database and have read access to a copy of the transaction database. In Maine the users maintain the transaction database and have read access to a copy of Boston database. Each site has a mirror copy of the other's database.

To keep the mirror databases in sync, Miles runs Silhouette in both directions betweeen the sites. Silhouette transmits name/address changes from Boston to Maine, where it applies them to a mirror copy of the database. This keeps Maine updated to within a few minutes of real-time. Silhouette also transmits transaction changes from Maine back to Boston, where they are updated to another mirror database. Each site has all the current information, has control of its own data, and provides emergency backup for the other site.

Applications such as spreadsheets, graphics and word processing are notorious consumers of CPU time. These benefit from being on their own dedicated computers. PCs are a good choice, as a dedicated PC often performs better than a busy Series 70 on CPU-intensive applications.

Word processing is another application which definitely should be on a PC. If you're running HPWORD or some other word processing package on your HP 3000, you're paying dearly for it. You should not allow any word processing on your HP 3000 unless it's dedicated to word processing. HP has been advocating this approach for a few years, and the development thrust of their software has been in this direction, with more PC-based software, and access software to upload and download the data. You don't necessarily need the latest and greatest integrated software for uploading and downloading. Reflection from Walker Richer & Quinn will do the job well.

Integrating PCs and 3000s.

Rolf Schleicher in Hamburg controls a network of over 200 PCs from his HP 3000, using Reflection and a few other tools. The 3000 automatically logs onto the PCs at night and backs up the hard disks for the users. He also updates the DBASE files on the PCs nightly with the latest information extracted from corporate IMAGE databases. He often uses his larger PCs as attached processors to expand the power of his 3000 (think of how many MIPS there are lying around unused at night!). For example, if Rolf has a large statistical analysis to do, he may download the data to a PC, start the analysis running and upload a single number later as the result. He finds that many system management tasks are easier to program in LOTUS than in COBOL. For example, managing disc space and file usage by doing :LISTF and :REPORT into a LOTUS spreadsheet for analysis.

A Company That Is Heavily Invested in PCs.

Compaq Computers has some 900 PCs in their company. Instead of downloading raw data files from the HP 3000, they have summary files lying around which they download using Reflection, feeding them into Lotus or graphics or whatever. They do all their graphics on the PCs except for one giant run of 85 graphs in DSG, which comes at month-end on the laser printer. It ties up an entire Series 70 until it's over. They don't attempt to do anything else on that machine until the graphs are printed. But all of their other graphics, what-if graphics, presentation graphics, is done on the PCs. This keeps the graphics hogs off the HP 3000s.

Upgrade to Faster Hardware — Tip #5

The Series 70 is a winner.

People we talk to say that their Series 70s are terrific, especially when they're loaded up with eight or nine megabytes of memory. They have a lot more horsepower than a Series 68. If you're on a Series 68 or smaller, you might consider going to a 70 instead of a 930 or 950. The Series 70 is so much more powerful than a 68 that we have heard that it is impacting the market for the Series 930.

When Longs Drugs upgraded one of their Series 68s to a Series 70, they went to U-MIT, Turbo IMAGE, and converted from Desk III to Desk IV all at the same time. At first they <u>didn't</u> see any difference in performance. But then they discovered that Desk IV ran 40% slower than Desk III! When they fell back to Desk III the system really took off! The extra power of the Series 70 masked the poor performance of Desk IV. Now that's horsepower, to be able to swallow up application problems as easily as that. When Boeing upgraded their TMS manufacturing machine from a 68 to a 70, they noticed a tremendous improvement in performance. Their 2-day backlog of batch jobs disappeared!

Use LAN/3000 instead of DS.

A LAN will <u>not</u> reduce your system load, but users report that it offers much higher throughput than DS with just about the same overhead. You have to replace an INP with a LANIC, and string coaxial cable instead of regular wires. Bill Gates at Longs Drugs says that for the small price of 3% more cpu, a job which was taking 50 minutes over a 56 kb line using DS now takes eight minutes over the LAN. When Northern Telecom in Lachine went from DS to LAN, they got more communication throughput without noticeable increase of cpu overhead. They have three Series 70s, a Series 52, and two Series 9000s connected together in the same room. Besides being faster than DS it costs less, because they require only one LANIC per machine instead of many INPs. Using a 'vampire tap' they can add another machine to an active communications wire without affecting any other machines.

7933XP drives with hardware cache seldom help and can actually hurt performance. A few sites have reported improved performance using 7933XP drives in place of MPE caching, but many more have not. Perhaps the Eagle XP drives will work better. They have 2 megabytes of cache space, are 20% faster, and have reduced the "pep" overhead to one millisecond per access (from 6 or 10 ms.). The new Falcon drives from EMC2 also show promise; they have 4 megabytes of cache, incorporate faster drives than the Eagles, provide caching for requests of up to 32K bytes (instead of 4K bytes), and claim to have a smarter cache algorithm.

More memory can help, unless you already have 3 megs for caching.

New CRTs with Forms Cache (2394) can improve response time.

New 9600-Baud Modems can make remote users smile. The Micocom AX/9624c modems understand HP's Enq/Ack protocol and have worked well on our Series 37 at Robelle. Remember, response time is perceived by the user, and a large part of that perception is not the processing efficiency of the programs, but the speed of the datacomm gear.

Successful Sites Discourage On-line Reports.

If you allow users unlimited access to run reports on the production machine, then why should we feel sorry for you? You are getting the slow response that you asked for. Reports should run in batch, because that is where you can control the total number at any one time.

Concurrent Batch Jobs.

On a Series 70, there is enough extra power to allow concurrent batch jobs. Some sites allow six or eight executing batch jobs at the same time. BOSE and Turbo Resources both restrict concurrent batch jobs to *different* user.accounts. See MBQ in the contributed library for ideas on how to control this.

The Night Time is the Right Time.

To ensure good response for on-line users, most of the successful sites we contacted had a policy of controlling the number of concurrent batch tasks allowed during the day. The 3000 will run just fine all night long, without anyone watching it. Many shops are shifting work from prime shift to graveyard.

What To Do When Overnight Jobs Don't Finish?

'Unfinished nightly jobs' is now a common complaint at HP shops, especially at month end (perhaps because people listened to advice to shift work to the evening hours). In our survey, we heard several methods for improving batch throughput: upgrade to a Series 70, get a separate CPU for reports, require department-head approval on job requests, reduce backup time, increase block sizes and, the most successful strategy, apply MR NOBUF tools wherever possible (as an HP SE said, "I have seen incredible speed improvement from front-ending QUIZ with SUPRTOOL. Software solutions to performance problems often show gains of 10 or 20 times. Hardware solutions, with no improvment in the efficiency of the underlying software, usually show gains of less than 1 or 2 times.")

Backup Taking Too Long.

Many people are spending 2 to 4 hours per night on backup. If you run out of night, there are ways to reduce backup time. Get high-speed tape drives. Look at BackPack from Tymlabs. HP's Copycat program and the FCOPY-FAST option of MPEX will do a high-speed disc-to-disc backup, after which you can let the users and jobs on again and do disc-to-tape backup at your leisure. Elbert Silbaugh at Boeing uses this method and keeps his system available 23.5 hours a day. Another Boeing site in our survey wrote a privileged program to copy the database disc-to-disc *while the users are still accessing it in read-only mode.* Their system is available 24 hours a day. (Adager can also copy a database while it is open for read-only.)

Use MR NOBUF Tools and Optimum Block Sizes — Tip #7

Problem. One of the most common destroyers of system performance is the notorious **serial scan.** When you copy an enormous file, or reorganize a KSAM file, or select 100 records to report with QUIZ by reading every entry in a million-record dataset, you are bogging down the computer. The default methods of doing a serial scan are extremely inefficient on the HP 3000.

Solution. One of the most impressive ways to speed up serial I/O is to use MR NOBUF (multi-record non-buffered, not Mister Nobuff). You can write your own code to take advantage of MR NOBUF access if you're careful, but you don't need to - you can purchase tools that do it for you. Popular tools which use MR NOBUF access are HP's DSCOPY (you can use DSCOPY for copying files to the same system), HP's COPYCAT for file copying and backup, MPEX's FCOPY/FAST and Tymlabs' COPYRITE for file copying and duplication (powerful for KSAM users). Robelle's SUPRTOOL does MR NOBUF serial file access for IMAGE datasets (and any other file type) and Running-Mate replaces serial dataset reads in applications.

The Power of MR NOBUF.

We got a call a while ago from a fellow who didn't even know he had SUPRTOOL on his system, because it came bundled with another package he had bought. He found it, and the documentation, on his system so he started using it. He had a QUIZ job which normally took two hours to run, cruising through a huge database. A total novice, using the instructions in the manual he used SUPRTOOL to front-end his QUIZ report. The total time for this daily job went from two hours down to 15 minutes.

One of the shops we interviewed still uses a service bureau for some big accounting merges in IBM batch. They're considering that if the Spectrum is big enough, they might use it for that. They used to have four service bureaus. Now they're down to one. They brought things in-house by giving them their own machines, finding packages like mailing-list software front-ended by SUPRTOOL.

Turbo Resources uses their HP 3000 to bill their credit card customers. At month-end, they had a batch program that generated 1,000,000 disc I/Os reading a 90 record control file. Sixty of the records were unnecessary and after reblocking the file, they were able to read it in one disc I/O. They now keep the control information in a table in memory, reducing one million disc I/Os to one.

Block Sizes.

The default blocking factors (number of records per physical disc block) is usually wrong. For big batch disc files, the maximum block size is now about 14K words (REC=14336), while the default is still the smallest block that will fit. The bigger the block, the faster the programs will run. For IMAGE databases the default block size is 512 words, as it has been since 1974. Many people we contacted in our survey were using 1024 words or more.

Compile Your Fourth-Generation Applications Tip #8

Problem. Interpreted Transact, and other 4GLs, consume too much CPU time.

Solution. Compile Transact source using the Fastran compiler.

When Cathy Vanderburgh was at Macmillan Blodel, she wrote up her experiences with Fastran as *Riding Herd on a CPU Hog*: "We recently developed a Transact system which included a large (15,000 lines) and complex (10 screens) data-entry program. After installation, the response times for the program varied from slow when the machine (an HP3000/64 with MPE IV) was lightly loaded to abysmal when the machine was heavily in use. Yet none of the other users on the system were experiencing similar problems at any time. We ran OPT/3000 to observe the execution of the program. The CPU time needed to interpret the IP code plus the complexity of the program was causing the MPE scheduler to class the process as a 'CPU hog' and to penalize it by dropping its execution priority. The only way to improve the response time would be to reduce the excessive CPU usage. Fortunately, this story has a happy ending. We discovered a piece of software called Fastran, a product of Performance Software Group, that compiles Transact source code into an executable program. On evaluation, we found that a Fastran version of the program used 1/4 to 1/3 of the CPU of the original Transact program, enough of a drop to bring the response back to an acceptable level. The user now enjoys(?) the same response patterns as everyone else on the machine. And the moral of the story? Without Fastran, of course, the author of the original program would now be busily re-writing it in COBOL. Plenty of programmers have discovered the hard way the functional limits of tools like Transact."

At CNR, where a large on-line application is written in Transact, compiling the application with Fastran led to a CPU reduction of over 60%, and a stack size reduction of 25%. Single-user elapsed run times did not improve much, but as more users were added, the reduced CPU requirements produced shorter elapsed run times. These numbers are for an I/O bound application where most of the time is spent in the database intrinsics and the file system; on CPU-intensive tasks the reduction can be considerably greater.

At Kitsap County they use Fastran over Transact wherever possible because the programs run faster. However, they have found a few cases that Fastran cannot handle. If a program needs extensive table handling, they choose COBOL over Transact.

Dexter Shoes was described earlier as running large manufacturing and distribution operation on a network of six Series 40s. Their entire application was coded from scratch in Transact. They report that this gives them the ability to respond to user suggestions in days instead of months. The reason they can get away with only Series 40s, instead of Series 68s or 70s, is that they compile the programs with Fastran.

Larry Kemp of HP Bellevue has found Fastran about 25% slower than COBOL and 50 to 98% faster than Transact (an 8 hour job reduced to 8 minutes was the best he ever saw!). An alternative 4GL that he found to give excellent performance is Protos; it generates a COBOL program for execution. And, finally, no one says you can't rewrite your most frequently used program in COBOL (use system logging to find out which program it is).

IMAGE provides calculated read, chained read, and serial read. OMNIDEX adds record selection across multiple fields, generic retrieval and sorted sequential access, multiple keys in masters, and keyword retrieval on text data. It does this by adding another structure to IMAGE's: the binary tree. Traversing this tree is fast, fast, fast.

HP uses OMNIDEX in the Response Center to index bug reports. That is how they can find out instantly who else has had a system failure 916 on Series 37 under T-MIT with a full moon. OMNIDEX indexes every word, not just the manually-assigned "keywords" as in the old SSB system. Doug Iles of HP says, "We could enter partial values and/or full values from several different fields and find 5 qualifying records out of 50,000 in seconds."

The people at D.I.S.C. (the suppliers of OMNIDEX) distinguish between "informational" data - data that you want on the system for doing inquiries, and "operational" data - data generated by the transactions of the organization. For example, in an order processing system, active orders are operational; customer and vendor master records are informational. Operational data is volatile and lightly indexed. Informational data is static and can afford to be highly indexed for fast, low-cost retrieval. In a general ledger system, the transaction dataset is operational. You do data entry and editing with it. When the transaction is completed, you post it to the ledger dataset, where it becomes informational data. You no longer modify it (much), but you need to ask numerous complex questions about it. OMNIDEX gives you the ability to index everything in your information data. You can use batch time to update the indeces, instead of on-line time.

Users also apply OMNIDEX to replace KSAM. The index-sequential part of OMNIDEX (called IMSAM) will reindex about 1 million keys per hour on a Series 70 (versus 20 to 30 hours with KSAM).

Example:

Kim Everingham at Consolidated Capital reports that they use OMNIDEX extensively in their tracking system for investors and investments. The power of OMNIDEX indexing allows their offical IMAGE structure to be very simple: masters for entities and details for transactions. They have 4.5 million sectors of data, 250 QUICK screens, 12-15 databases, and 35-40 users on a Series 70. Without OMNIDEX the application would require an IBM mainframe. Within 1 or 2 seconds they can identify an investor and the investments he is involved with, even if the investor only gives a vague or partial description of himself (e.g., trust company, Ralph, Minneapolis). They do all updates on-line, including updates of the OMNIDEX indeces; the only exception is the entry of new investments -- that is done in a nightly batch job due to the serious impact on response. They have plenty of horsepower with the Series 70; the only bottleneck is that QUICK consumes about 60% of the CPU time, but this hasn't impacted response time yet. They also use SUPRTOOL for ad-hoc extracts and as a QUIZ front-end.

Kitsap County Government is an HP site that gets a lot of work done without hitting the limits of the HP 3000 line. Jim Kellam, the manager, started with a Series 48, overloaded it, then added a Series 68 and left the 48 for development. He reports that OMNIDEX inquiries are unbelievably fast ('find all the voters named Smith' instantly replies '1200 entries found'), but can be abused, just like any tool. For example, one of their programs opens all eight databases at the start, in case you might need them. Installing OMNIDEX implies an extra open and another extra data segment, the equivalent of 16 DBOPENs per user. The users sometimes get in and out of the application to access other software, so they pay this startup overhead more than once per day. The IMSAM part of OMNIDEX allows you to define concatenated keys with pieces from 3 different datasets. Jim feels that they may have overused these features, because he observes slow response with some of these bizarre keys.

Computer Insecticide

The Art of Debugging

By David J. Greer

Robelle Consulting Ltd.
8648 Armstrong Rd. R.R. No. 6
Langley, B.C. Canada V3A 4P9
Telephone: (604) 888-3666
Telex: 04-352848

Copyright Robelle Consulting Ltd. 1987.

Permission is granted to reprint this document (but not for profit), provided that copyright notice is given.

Computer Insecticide

The Art of Debugging

By David J. Greer

If you asked me for the primary difference between a programmer right out of college and one with five years' experience, I would say "the ability to debug programs". Debugging is still an art, not a science; but there are some techniques that can help. This paper provides guidelines and suggestions for finding and solving bugs. *Every* reader should find at least one new debugging idea.

Personal Background

I am responsible for all of the programming and documentation of two of Robelle's four products: SUPRTOOL and XPRESS. Both are large programs written in SPL. SUPRTOOL is a batch optimizing tool and most of the code is technical in nature. XPRESS is an electronic mail package that uses an IMAGE database for its data structures. The XPRESS code is more application oriented. I also work on Prose, our text formatter, which is written in Pascal, and have done a great deal of application programming in COBOL. I haven't done much work with Fourth Generation Languages.

Computers Have Always Had Bugs

As long as there have been computers, there have been bugs. You have probably heard the story, attributed to Grace Hopper, of the first "bug". According to the story, the programmers on one of the first digital computers were having great difficulty getting a program to work. One time it would fail -- the next time it would succeed. After numerous fruitless revisions to the program, someone happened to look inside the cabinet. An insect had gotten into the vacuum tubes and relays and been zapped. It was acting as an intermittent connector, changing the wiring of the computer from time to time. They removed the bug and the program worked. If only all bugs were so easy to find and remove.

Organization of this Paper

We will present our ideas on debugging in five steps:

1. An Example: debugging is easier to show than it is to discuss.

2. Search Techniques: where in all those thousands of lines of code is the bug.

3. Testing Techniques: there are several areas where you should concentrate your efforts.

4. Development Environment: tools and techniques that help programmers write, test, and debug faster.

5. A Final Example: another example of how debugging principles were used on a real-life bug.

An Example

If something can go wrong,
it will.

Murphy's Law

A user reported the following problem in our SUPRTOOL product. When you exitted from SUPRTOOL, the following error message was displayed:

```
Warning:  Using DBGET for the input records

IMAGE ERROR AT %001142: CONDITION WORD = -11
DBCLOSE, MODE 3, ON #2 OF <NULL>
BAD DATA BASE REFERENCE (FIRST 2 CHARACTERS)

ERROR: Unable to rewind dataset with DBCLOSE
```

Search Techniques

To solve this problem, we need to know the sequence of events that led to this message. Before this error message appeared, these commands were entered into SUPRTOOL:

```
>BASE TEST,3          {Open the database in mode-3}
>GET DLINE            {Request input from the dataset DLINE}
>BASE                 {Close the database}
>EXIT                 {Exit from SUPRTOOL}
```

Isolating The Problem

There are approximately 35,000 lines of code in SUPRTOOL. There is only one place where DBCLOSE-Mode 3 is called. We discovered this by searching for the string "DBCLOSE" in all of the SUPRTOOL source files. SUPRTOOL is written in small and modular pieces and there are only three places where DBCLOSE is called. Only one of these uses mode-3. Given the sequence of SUPRTOOL commands, the procedure where the DBCLOSE-Mode 3 call is located should *never* be called.

This piece of code should never have been executed, but the evidence says that it was. Reading the code reveals the following if statement:

```
if in'filenum <> 0 then
   call code where DBCLOSE-Mode 3 fails
```

At this point, the sequence of SUPRTOOL commands guarantees that in'filenum *must* be zero. The SUPRTOOL command:

```
>BASE
```

is supposed to guarantee that the database is closed and in'filenum is set to zero. Close examination of the code showed that for the special case where the database was open in mode-3, the in'filenum was not being reset to zero. We have found the bug.

Software Engineering

While isolating the code, we had two "guarantees". They were:

1. The call to DBCLOSE-Mode 3 should never have been executed for the command sequence entered.

2. Given the command sequence, the in'filenum variable had to be zero.

These "guarantees" are called assertions. An assertion is something that *must* be true in a given circumstance. SUPRTOOL is writtences. This makes it easier to find and verify the assertions.

Conclusion

This example shows the three main points that we want to make about debugging:

We used *search techniques* to isolate the problem. When we were first presented with this problem, we did not have all of the command sequence.

We used *testing techniques* to check our assertions (e.g., we first opened the database in mode-1 which worked fine).

Using the tools in our *development environment*, it was possible to isolate this problem to one line of code out of 30,000.

Search Techniques

When you have eliminated the impossible,
whatever remains, however improbable,
must be the truth.

Sherlock Holmes in *The Sign of Four*

Before fixing software problems, we need to find them. In this section, we will suggest techniques for reviewing both code and data to find the one piece of code or data where the problem occurred.

Reproducing The Problem

Start by getting the *exact* input if you can. Try reproducing the problem in your test environment with as little data as possible. Ask about all of the conditions present when the problem occurred (e.g., other users, file equations, batch jobs, etc.).

Sequential Search

The simpliest way to find the problem is to start at the beginning and read to the end. You start with the first line of code in your program and you follow the logic one line at a time. Study your database by listing it sequentially. With small programs and small amounts of data this is effective.

The Game of Clue and Code

Parker Brothers produces a popular game called **Clue**. The objective of **Clue** is to deduce the solution to a crime by a process of elimination. A game might end with, "Colonel Mustard did it in the bedroom with the wrench". With programs, you can do the same thing by doing numerous, carefully selected test runs, each of which changes only one factor. From the differences in the results, you can often deduce exactly which module the error is in and even which data structure is involved. The SUPRTOOL example used the "clue" method.

Let your program execution give you clues to the problem areas. As early as possible eliminate as much code as possible. Try to find the assertions, these usually provide the best clues. Check boundary conditions. Many problems arise around boundary conditions. Look for "boundary" clues. For example, in our DBAUDIT program had a bug in it recently that was a classic case of a boundary condition. A detail dataset with 9 or more critical fields would cause DBAUDIT to produce unpredicatable results.

The Game of Clue and Data

A typical problem with production application systems is that something goes wrong with the data, but you only find out long after the problem occurred. Playing **Clue** in these cases involves eliminating as much data as possible.

Even if you know that one of 300 records is wrong, it is usually impossible to examine all 300 records. Nor is it possible to trace your program execution through all 300 records. Use all of your information to reduce the amount of data.

Use IMAGE paths to isolate a subset of your dataset. If you have a transaction file with a million records, try to reduce the data to a specific period. For example, start with the data for one month, then one week, and finally one day. It is easier to debug if you can reproduce the problem with *one* record from your database.

Data Assertions

Assertions about your data are just as important as assertions about your code. Clues often appear with inconsistent data. Is there a transaction with an amount less than zero (when you know that all transactions must have an amount greater than zero)? Do you have date fields? Write a program that checks whether all date fields contain a valid date. If a record has an invalid date, it may contain other invalid information.

Tracing Execution

When looking for clues, it is often difficult to answer the question "did statement-x get executed". Typically, we resort to inserting tracing code in our programs, or use TOOLSET to do it for us. Many programmers add a display statement for every SECTION in their COBOL program. This is usually the slowest way to trace code because there is too much output.

Assertions are the best way to trace your code. Add DISPLAY statements to check your assertions. Often, a half-dozen display statements will give you enough clues to find the bug.

Use assertions to choose which variables to display. In the SUPRTOOL example it was not necessary to add tracing code because we knew that the problem was caused by in'filenum not being zero. If we had further difficulty in isolating the code, we would have added tracing code to print the value of in'filenum in selected procedures. Note that we would not have printed all 200 SUPRTOOL global variables.

Problem Classification

The causes of programming bugs vary in their size and complexity. We would classify problems into three general categories:

1. One-line problems. This is where a few lines of code are incorrect. One of the most common one-line problems is a boundary condition (less than instead of less than or equal). Many of these problems can be discovered early if you watch for off-by-one errors and if you test programs at their boundary values.

2. Invalid data structure: you have chosen the wrong data structure. Problems with the current record in IMAGE are one example of this class. Fixing these problems is usually more difficult than fixing one-line problems. Common data structure errors include using too small a variable to hold a total value. If you are totalling a PIC S9(4) COMP variable, you should use PIC S9(9) COMP for the total variable.

3. Incorrect algorithms: the algorithm does not solve the problem (you only thought that it did). For example, erasing a master dataset.

With IMAGE, a change in a data structure often requires changes in the algorithm that accesses the data structure. For example, you currently use a master dataset, but you need to change it to a detail dataset and add an automatic master to provide access by two key values. This requires a change to the data structure (changing the master dataset to a detail) and a change to every program that accesses the master dataset (changing a DBGET-Mode 7 to a DBFIND and DBGET-Mode 5).

Examine All of the Evidence

This includes the input data, output data, the source code, any library routines, and anything else that might help (even the documentation). Are there any file equations in effect (use :LISTEQ)? Are there any temporary files (use :LISTFTEMP)? Check your premises before you invest too many hours.

Check that you are using the right SL file (use LMAP for this) and that the subroutines in the SL file are correct. See if you are linked to an old SPL subroutine that you didn't know about. Check the obvious: is your program using the correct database?

Keep An Open Mind

If you think you've identified the section of the program that contains the bug, but there appears to be nothing wrong with it, look somewhere else. Showing the code to another person can highlight the problems with those pieces of code "that couldn't have a bug". If you have a really bad bug, leave it overnight. Often, the answer will be sitting there in the morning.

If you cannot reproduce the bug, examine the source code for problems. Often you will discover incorrect code, but sometimes this is not the code that caused the original bug. Don't stop, you will usually find the problem.

Summary

Searching is a process of elimination.
Check data assertions.
Trace execution with DISPLAY statements or TOOLSET.
Three types of bugs: one-line, bad data, and bad algorithm.
Examine all the evidence.
Keep an open mind.

Testing Techniques

Keep it as simple as possible,
but no simpler.

Albert Einstein

We assume that you would like to find the bugs *before* they happen. Our current knowledge of software engineering does not guarantee that bugs will be absent from computer programs. Our only solution is to test our software for the absence of bugs.

Keep It Simple Stupid - KISS

Writing complicated software is an open invitation for Murphy to descend. Writing and debugging code is already difficult; don't make it worse by inventing tricky data structures or fancy algorithms. Use the simplest idea that will work. Optimize later, only when you *know* that the optimization is necessary.

Boundary Conditions

Errors are most frequent on boundary conditions. For example, beginning of file, end of file, empty file, full file, beginning of loop, end of loop, entry to module, exit from module, value less than limit (instead of less than or equal), table overflow, or table empty. When verifying code, check that the boundary conditions are what you expect.

Test Assertions

Check each assertion. You should take some time to create errors in your program to test error conditions. How do you know that your fatal IMAGE processing is correct if you have never tested it?

Write and Test Small Pieces

We typically write code in 50-100 line increments. This means that every 50-100 lines of code are compiled, run, and tested. We verify that each small piece of code is working *before* proceeding to the next piece of code. Later, we only have to test the interfaces between each small procedure.

Test each module (a module is 1000-2000 lines of code) as it is completed. To complete testing, check that the various modules interact with each other. In the typical COBOL/IMAGE environment, procedures become SECTIONs (they should be less than 100 lines of code) and modules become programs or subprograms.

System Testing

This testing involves a methodical approach to the data and careful testing of different modules. You should choose a small, but representative group of data. This data should cover all of the common cases. Include test data for exceptional conditions that you expect in your application.

The less manual testing, the better. Automate your test environment using job streams. At Robelle, we test SUPRTOOL with over 300 tests organized into 35 job streams.

Have good control over your test environment. Store a copy of your test data. Assume that programs you will be testing will destroy your test data.

The SUPRTOOL Test Environment

The SUPRTOOL test jobs are organized into 35 job streams. We carefully selected a few data files for use in these job streams. We also wrote a set of programs to help in verifying the execution of SUPRTOOL. Here is an example of an actual SUPRTOOL test:

```
!job jtest02,bob.green,suprtest;outclass=lp,3,1;inpri=7
!comment
!comment  setup:      12 May87 by David Greer
!comment
!comment  The next test checks that when suprtool is :RUN
!comment  with PARM=16, the file INPUT is copied to the
!comment  file OUTPUT.
!comment
!purge file1x
!file input=file1
!file output=file1x
!run sttest.pubnew.robelle;parm=16
!run compare
file1
file1x
!purge file1x
!comment
!comment  End of JTEST02.
!comment
!run result;parm=2;info="PARM="
!set stdlist=delete
!eoj
```

SUPRTOOL Test Description

To reduce the information from these job streams we use the following techniques:

1. The $STDLIST listing uses a low outclass priority (3). We almost never print the test job $STDLIST listings.

2. If anything goes wrong in the job stream, it is aborted by setting the fatal JCW.

3. The COMPARE program compares two files and sets the fatal JCW if they are not identical.

4. The RESULT program sends a :TELL message describing which job has completed. It also appends a record to the file RESULTD.SUPRTEST. A listing of this file show which jobs completed.

5. If the job stream completes successfully, the $STDLIST listing is deleted.

If a test fails, the job stream listing is not deleted. This lets us examine the listing to see where the error occurred. After the tests have completed we have to only list the result file to see our test results.

Once a test-bed environment has been set up, it becomes easier to add new tests. Every time we find a bug in SUPRTOOL, we attempt to devise a test that will discover the bug if it every shows up again. It is common for a bug to reappear in later versions of a program.

Scaffolding

In the *Mythical Man-Month*,Hscaffolding as "the programs and
data that are used for debugging, but which never appear in the final application". Brooks also suggests that "it is not unreasonable for there to be half as much code in scaffolding as there is in the product". The COMPARE and RESULT programs from the SUPRTOOL test job are examples of scaffolding. A test database is another example. If you intend to build reliable software, be prepared to invest resources in building the scaffolding.

Pascal Validation Suite

The Pascal Validation Suite is a set of programs to test a Pascal environment (compiler, linker, loader, and run-time environment). This suite of programs has been used to test many Pascal compilers, including HP's and our own, for compliance with the international standard for the language. The suite designers distinguished between two classes of tests: conformance and deviation.

Conformance

A conformance test attempts to verify that a program will execute in a given set of circumstances. For example, you may accept a transaction amount from $0.00 to $9,999. A conformance test would insure that the program accepted any amount in the specified range. The SUPRTOOL test above is a conformance test.

Deviation

A deviation test attempts to make a program fail in a given set of circumstances. Note that a deviation test discovers a bug if the program executes without an error. For the transaction amounts above, deviation tests would verify that the program produced an error for amounts less than zero, greater than $9,999, or if three decimal points were entered.

The Pascal Validation Suite contains many deviation tests. The following one should be understandable, even to non-Pascal programmers:

```
{TEST  6.1.5-4, CLASS=DEVIANCE}

{The number productions specified in the Pascal standard
 clearly state that a decimal point must be followed by a
 digit sequence.  The [Pascal] processor deviates if the
 program is acceptable, in which case it will print
 'deviates'.  The processor conforms if the program is
 rejected.
}

program t6p1p5d4(output);
var
   i : real;
begin
   i := 0123.;
   writeln(' The value of I is ', i);
   writeln(' Deviates...6.1.5-4, number syntax')
end.
```

The compiler should produce an error when it compiles this test program. If it compiles and runs the program, then the deviation test has failed.

Note that like the SUPRTOOL test, the Pascal test is self-identifying. The test suite is organized around the ISO standard document for Pascal (e.g., 6.1.5-4 is the fourth test of section 6.1.5 of the Pascal standard).

In general, it is easier to perform conformance tests. When automating deviation tests, care must be taken to verify that the program stopped execution because of the deviation error and not for some other reason.

Summmary

Keep your software simple.
Explicitly test for boundary conditions.
Write code to check assertions.
Write and test code in small pieces.
Use automatic testing wherever possible.
Invest in scaffolding to assist debugging and testing.

Development Environment

It's the little things that count.

Greer's Law

Excellent carpenters use the best tools. Good programmers should also be given the best tools. Program development follows this algorithm:

```
while not problem-solved do
  while bugs-still-exist do
  begin
    write code
    compile code
    test code
    verify results
  end
```

Do *your* tools help each statement of this process?

Writing Code - The Programmer Environment

An editor is the programmer's most important tool. If you accept the algorithm presented above, you will expect your text editor to provide facilities for writing code, compiling, prepping, and running programs. Every time you must exit from your editor, you lose at least ten seconds (ignorning the time for /Text and /Keep). Worse, it's a thorn in the side of each of your programmers.

QEDIT provides all of the facilities for creating, compiling, prepping, and testing code. It is optimized to make the most common operation, writing code, as fast as possible. If you can't afford QEDIT, try QUAD from the contributed library.

Verifying Program Execution

QUERY has added immensely to the power of IMAGE. With QUERY it is possible to verify execution of a program which modifies an IMAGE database. If you use MPE files or KSAM files in your application, you must write a program for *every* file which prints out the contents in a readable way. Using the FCOPY or SUPRTOOL OCTAL,CHAR listing on MPE or KSAM files is asking for trouble.

QUERY has two potential problems. QUERY is very slow at sequentially reading a large dataset. You cannot hold QUERY as a son process from your editing environment. Using process suspension, you should be able to switch between your editing environment and your verifying environment in under a second.

SUPRTOOL solves both of these problems. First, it is very fast at sequentially reading an entire dataset. SUPRTOOL includes a database editing package especially designed

for programmers verifying program execution. Finally, SUPRTOOL can be held as a son process with the database open. Switching context between editing and verifying is almost instantaneous.

If you have MPEX from VESOFT, you can also have fast context switching. You can hold QUAD as a son process and do your compiling, prepping, and running from within MPEX. You can also use MPEX to quickly examine MPE and KSAM files.

DBAUDIT and Program Verification

Transaction logging is an optional feature of IMAGE that transcribes all database changes to a logfile on disc or tape. DBAUDIT will play back IMAGE logfiles, showing you what values were added, deleted, and modified.

Logging and DBAUDIT give you another perspective on your applications. You can verify program execution by using logging what programs are actually doing to the database. If the programmer who created the program has left, this may be the fastest way to find out what the program is trying to do. If you don't have any money in your budget for DBAUDIT, start by trying LOGLIST from the contributed library.

Suppose you have acquired an Accounts Payable package from an outside software vendor and some functions of it are not working properly. How do you report your problems to the vendor with enough information to ensure that he will be able to correct the bugs?

One way is to turn IMAGE logging on and then print out the database transactions that you suspect may not be working properly. This record of what the programs actually did to the database may contain just the hard facts that the vendor will need to fix the errors.

MPE Accounting Structure

Take time to implement your MPE accounting structure. A poor choice hinders development and is difficult to change after the fact. Avoid moving users or programmers around from account to account or group to group. All of those :HELLO commands will start slowing your machine down. Here is one suggestion for an MPE accounting structure.

Have three different accounts. One for development (e.g., DEV), one for testing (e.g., TEST), and one for production (e.g., PROD). The group structure within each account should be identical. Some group suggestions might be:

COMPILE

This group contains should contain one file per program. Each file should be an MPE :STREAM file which will recompile the program the program with the name of the file in the group. You may wish to only create job streams for programs that consist of more than one source file, otherwise use MPEX from VESOFT.

DATA

Put all IMAGE databases, all KSAM files, and all MPE files in this group. You should include the schemas that created the database, but be sure to remove the passwords first.

PUB

This group should contain only program files and one SL file.

SOURCE

The source code for each program. The file-naming scheme you use should be flexible enough so that programs which consist of multiple source files will have similar filenames.

Program Identification and Version Control

Every program should have a name and a version number. Reporting programs should show the program name and the version number as part of the heading line. Find room on your V/PLUS forms for the program name and the version number.

Whenever you make *any* changes to a program, increment the version number. Keep the version number up-to-date on all related documentation. Use the version number to control installation of new versions of software into production.

Moving Programs

Programs should be modified in the DEV account. When a program is released for programming, the following steps should be taken:

1. Move the source code to the TEST account.

2. Move the COMPILE job stream to the TEST account.

3. Purge the program file from the PUB group of the DEV account.

4. Use the COMPILE job stream to recompile the program in the TEST account.

5. Test program execution against the test data.

6. If there are no problems, move all files from the TEST account to the PROD account.

If any bugs are found, do not allow changes to be made in the TEST account. Copy the source code back to the DEV account, where it is repaired and tested by the programmer. Then repeat the steps to move the program back to the TEST account.

Naming Conventions

Just as naming conventions are important for programs and databases, they are also important for the files in your applications. The MPE file system makes it difficult to choose meaningful filenames. One solution is to pick arbitrary filenames and keep an index of what every name means (e.g., MIS001). Another solution is to use added group names (e.g., BUDGET01.REPORTS.ACCTING).

Files that logically belong to the same program should have the same filename in each of the different groups: BUDGET01.DOC, BUDGET01.SOURCE, BUDGET01.PUB, BUDGET01.COMPILE.

Summary

Get a powerful text editor.
Verify program execution with QUERY or SUPRTOOL.
Use transaction logging to monitor execution.
Build a rich MPE accounts structure.
Enforce version controls and naming standards.

A Final Example

Inside every large program is a small program struggling to get out.

Hoare's Law of Large Programs

Every command-driven Robelle product (QEDIT, SUPRTOOL, and DBAUDIT) has a calculator command. This command is implemented by calling a standard calculator subroutine. This is a good example of modular program development. All three products have a calculator, but there are only ten to fifteen lines of code in each product to implement the calculator. One problem with this approach is that a bug in the calculator routine shows up as a bug in *all* three products. The following is a description of one bug that showed up in three different ways.

The Original Problem

The calculator takes an *expression*, evaluates it, and prints the result. Typical *expressions* would be:

```
=20+15                  {add two numbers together}
Result= 35.0
=20*15                  {multiply the same numbers}
Result= 300.0
=1e50*1e50              {computation overflow}

ERROR:  Overflow of your calculation, result is invalid

Result= .0
```

Care was taken to insure that any calculation overflow, underflow, or division by zero was correctly reported. We wanted to be sure that the calculator routine would not abort if one of these exceptional conditions occurred. A trap routine was written in SPL to catch and report these errors.

```
procedure calc'aritrap(long'result,trap'type);
   value   trap'type;
   integer trap'type;
   long    long'result;
   option internal;
begin

  if trap'type.(10:1) = 1 then    <<floating overflow>>
  begin
     p "Overflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
  end'if
  else
  if trap'type.(9:1) = 1 then    <<floating underflow>>
```

```
  begin
     p"Underflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
  end'if
  else
  if trap'type.(8:1) = 1 then     <<floating divide by zero>>
  begin
     p "Division by zero attempted, result is invalid"err;
     long'result := 0.0L0;
  end'if;

end'proc;    <<calc'aritrap>>
```

The calculator will display the result in three different formats. The default is to print the result as a real number. The other three formats are Octal, Double, and Bit. To produce each of these results, the long-real result is converted to a double integer (PIC S9(9) COMP). For example:

```
      =10,b                {the 16-bits in the number 10}
      Result= %(2)00000000 00001010
```

A Wrong Assertion

If you examine the code for the calc'aritrap routine you will find a simple assertion. The arithmetic errors that can occur are floating overflow, underflow, and divide by zero. What happens if you convert a huge long-real number to a double-integer? In all of our products you were aborted with an integer overflow error:

```
      =1e50,b              {convert a large number to double}

      ABORT :SUPRTOOL.PUB.ROBELLE.%0.%1230
      PROGRAM ERROR #1 :INTEGER OVERFLOW
```

Our first attempt to fix this problem was to isolate all of the code that converted long-real values to double-integer into one subroutine. We would check for overflow in this subroutine.

```
double subroutine longtodouble(long'value);
   value long'value;
   long long'value;
begin
   longtodouble := fixr(real(long'value));
   if overflow then
   begin
      p"Overflow of your calculation, result is invalid"err;
      longtodouble := 0d;
   end'if;
end'subr; <<longtodouble>>
```

Our Second Assertion

There is a very subtle assertion in this subroutine. It assumes that the 'if overflow then' statement will be executed. Because traps are enabled, this assertion is false. Before the if statement is executed the calculator has aborted with an integer overflow.

Even worse, we never tested this piece of code. It was so obvious that we knew it *must* work. Of course, we were wrong and the calculator would still abort with integer overflow.

One More Solution

Another problem with this solution is that it assumes that the next person to enhance the code would remember to do all long-real to double-integer conversions by calling the subroutine. The chances are that future enhancements would not use the subroutine and new integer overflow bugs would be introduced.

We already have a mechanism for detecting integer overflow errors. We have our original calc'aritrap routine. It seemed to make more sense to modify it to process integer overflow routines. Our existing calc'aritrap routine only handled long-real problems. Now it must handle both long-real and double-integer problems. Many COBOL programmers may find this code difficult to understand, but it shows our attempt at catching overflow errors.

```
procedure calc'aritrap(trap'type);
   value   trap'type;
   integer trap'type;
   option internal;
begin
  long pointer long'result = q-5;
  double       dbl'result  = q-6;

  if trap'type.(11:1) = 1 then    <<integer overflow>>
  begin
     p "Overflow of your calculation, result is invalid"err;
     dbl'result := 0d;
  end'if
  else
  if trap'type.(10:1) = 1 then    <<floating overflow>>
  begin
     p "Overflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
  end'if
  else
  if trap'type.(9:1) = 1 then    <<floating underflow>>
  begin
     p"Underflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
  end'if
  else
  if trap'type.(8:1) = 1 then    <<floating divide by zero>>
  begin
```

```
    p "Division by zero attempted, result is invalid"err;
    long'result := 0.0L0;
  end'if;

end'proc;   <<calc'aritrap>>
```

This time we did test the calculator for integer overflow. Everything worked great so we installed this version of the calculator in all of our products. After about a year, we received a telex with the following example:

```
=1e50*1e50          {computation overflow}

ERROR:  Overflow of your calculation, result is invalid

ABORT :SUPRTOOL.PUB.ROBELLE.%0.%3347
PROGRAM ERROR #24 :BOUNDS VIOLATION
```

Unbelievable, but after changing the calc'aritrap routine we forgot to test for long-real overflow. Our single biggest problem was that we did not have any well-established tests for the calculator. We have now solved the problem with the following job stream:

```
!job jtest01,bob.green,suprtest;outclass=lp,3,1;inpri=7
!comment
!comment  setup:      12 May87 by David Greer
!comment  purpose:    This job stream tests basic command
!comment              invocation within suprtool (including
!comment              the calculator).
!comment
!comment
!comment  test calculator
!comment
!purge file1x
!run sttest.pubnew.robelle
=10+32
=10-32
=10*32
=10/32
=10**32
=10+32,b
=10+32,d
=10+32,o
=%10
=%10,o
=%10,b
=%10,d
=%10 %10
=%10 %10,o
=%10 %10,b
=%10 %10,d
=-1 -1,o
=-1 -1,b
=-1 -1,d
```

```
=1e50*1e50
=1e-50*1e-50
=10/0.0
=1e50,d
in file1
out file1x
exit
!run compare.suprtest
file1
file1x
!purge file1x
!run result;parm=1;info="Calculator"
!set stdlist=delete
!eoj
```

Note that we do not check the results of the calculator. We have never had a bug with the actual results of the calculator, but we have had no end of problems with overflows. Note that the trap routine is a boundary condition. It is never invoked unless the calculator is stretched to its limits.

What about our final calc'aritrap routine? Most of you will never need an arithmetic trap routine, but this one works for double-integer overflow and long-real overflow, underflow, and divide by zero. Of course, it still has a bug or two: it won't handle single-integer overflow.

```
procedure calc'aritrap(trap'type);
   value   trap'type;
   integer trap'type;
   option internal;
begin
  long pointer long'result = q-5;
  double       dbl'result  = q-6;

  if trap'type.(11:1) = 1 then    <<integer overflow>>
  begin
     p "Overflow of your calculation, result is invalid"err;
     dbl'result := 0d;
     return 1;
  end'if
  else
  if trap'type.(10:1) = 1 then    <<floating overflow>>
  begin
     p "Overflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
     return 2;
  end'if
  else
  if trap'type.(9:1) = 1 then    <<floating underflow>>
  begin
     p"Underflow of your calculation, result is invalid"err;
     long'result := 0.0L0;
     return 2;
```

```
  end'if
  else
  if trap'type.(8:1) = 1 then    <<floating divide by zero>>
  begin
     p "Division by zero attempted, result is invalid" err;
     long'result := 0.0L0;
     return 2;
  end'if;

end'proc;   <<calc'aritrap>>
```

Conclusion

Our current knowledge of software development does not allow us to completely eliminate bugs from our code. This paper tries to show areas where we, as software developers, commonly leave gaps for bugs to creep through. Good luck with your software and may you never have to search for bugs.

Debugging Checklist

Search Techniques
Sequential and binary search are weak methods.
Clue: a process of logical eliminiation.
Checking assertions and tracing execution.
Examine all of the evidence and keep an open mind.

Testing Techniques
Test boundary conditions and assertions.
Write and test small chunks of code.
Automate the testing of your final system.

Development Environment
Use an editor that compiles, preps, and runs.
Verify execution with QUERY/SUPRTOOL, DBAUDIT/LOGLIST.
Take advantage of MPE's account structure.
Enforce version control and naming standards.

ABSTRACT

Overview of HP's New Network Products

Bernard Guidon, Hewlett-Packard Company

In this presentation the new HP networking solutions and products introduced since the Detroit Interex conference will be discussed. Included in this overview will be HP's Private X.25 Network, StarLAN, and Network Management products. Guidon will focus on the key features that HP offers in its network product line, including: Conformance to emerging international standards; Multivendor connectivity; Flexibility to grow and change to meet customers' needs and; A variety of WAN and LAN network solutions.

A schedule highlighting the network presentations at Interex will be available at this talk.

EFFECTIVE INFORMATION NETWORKS

COMBINE FLEXIBILITY WITH SECURITY

by Howard Gunn

Vice President of Marketing

Gandalf Technologies Inc.

The development of distributed information networks is a growing trend in all areas of business, industry, education and government. The evolution is driven by the major benefits of such networks. For simplicity, we can assume that the primary benefits are greater productivity and a sharper competitive edge through timely transmittal of time-sensitive information.

However, the proliferation of distributed information networks also increases the possibility of sensitive data files being used incorrectly, being fraudulently manipulated or even maliciously damaged and destroyed. Network managers must take a series of precautionary steps to provide the best possible protection against unauthorized network penetration. This

paper examines the trade-offs between speed, timeliness and flexibility on one hand and security problems on the other. It concludes that network managers must take specific actions to insure that physical and logical security is achieved and that other precautionary actions may be necessary. Furthermore, the protection strategy may even impact the communication architectures being deployed in the networks that are being protected.

Why Information Networks

The personal computer (PC) is perhaps the symbol of the modern office; it, the minicomputer and shared user micro have had a profound effect on data processing operations. The growing popularity of these computers, coupled with the availability of powerful software tools, has resulted in an explosion of stand-alone data processing units in many different departments of organizations throughout the world.

But the computers in these separate department or business units normally function only as individual entities. Data generated by staff using the

computers often remains within their personal orbit and is not readily available for profitable use by other personnel in the same department, much less in other departments.

This restriction on the availability of data or information (processed data) can have substantial negative impact on the effective operation of an organization. At best, it leads to unnecessary and costly work duplication; at worst, important work never gets carried through to a fruitful conclusion. For example, marketing data may be continually rekeyed, regenerated or reformatted by different departments by people who are unaware of or unable to access associated data in other locations. Or, business unit staffs may not even launch productive projects, if they are unaware that the data required for implementation is available in some other department.

Impediments such as these are now being solved through the development of information networks. These networks promote information exchange by interconnecting PCs, departmental minis, and shared user micros with other

corporate computers, databases and their associated terminals, printers and other devices.

Ideally, the result can be one cohesive network in which authorized staff can speedily and efficiently access computers, other system resources and application software from any terminal device, regardless of whether they are sitting in the same room, building, city or even country as the host computer. But, in developing this total information network flexibility, it is necessary to seriously consider how data network security is implemented and how it should be enhanced to insure information protection in a distributed network.

The Security Problem

Protection of data and information from unauthorized access or use is the basic security problem for the MIS manager. Although the unauthorized use of information was a problem before computers were invented, their advent has certainly not slowed mankind's desire to gain an unfair advantage from

timely information. And, in some ways, computers, databases and application software have made it easier.

As information networks grow in size, the security problems expands. Information networks bring into play a growing number of computers, terminals and other devices with perhaps thousands of access points with potential for unauthorized entry. In addition, the networks are oftentimes designed to permit public network access over phone lines. These public accesses have often used low level security procedures because individuals using the public network wanted fast access to information without resorting to complicated sign-on routines.

Computer hackers are perhaps the most well known example of an outsider gaining unauthorized access to computers. The movie "War Games" was a chilling example of their impact. Normally lacking criminal intent, these amateur computer buffs simply want to prove that they can "crack" a system. However, in the process, they can cause significant damage to or even destruction of valuable computer files. And there is no way of telling how

much real-time is consumed just defending against the hacker.

Another type of outsider is the person who attempts to gain unauthorized access to a computer for fraudulent purposes. These purposes might include manipulating computer files to credit bank accounts, creating false payments or tapping proprietary data, such as sales results or customer lists, for resale to competitors. Embezzlement by phone could one day be the easiest crime to commit.

These external threats are significant sources of security problems. But, internal threats in the form of dishonest or disgruntled employees who seek unauthorized access to sensitive and restricted information are even more important.

Dishonest employees, for example, may try to obtain customer lists or other computer files for their own advantage. Disgruntled employees have been known to gain entry to computers to delete or even destroy information vital to the existence of the organization.

I know of an example where a fired programmer inserted an internal loop in a Materials Requirement Planning (MRP) application that took his replacement seven weeks to correct. During that interval, the automated procurement processes tripled the inventory of raw goods for the company.

Consequently, there is a strong need to achieve effective protection against unauthorized access to computer files in an expanding network environment. This need can be better met if the management of an organization recognizes the gravity of the situation and develops a basic understanding of how to solve network security problems.

Addressing Network Security Issues

The most effective way to solve network security issues is to build in protection against unauthorized access so that it is an integral part of the network process and operation. This kind of security can best be achieved by establishing a number of protective barriers or security levels

between computers, data files, application software and users. But, to understand how these network security levels function, it is first necessary to discuss security measures in a general context.

For practical purposes, there are five general levels of security necessary to deal with the complexity of a distributed network system. They are:

1. Physical
2. Logical
3. Computer
4. Application
5. Access

Physical security deals with the relationship of a physical terminal location in the environment and how it is secured. Logical security deals with the terminal's ability and functionality specified by the network manager to the terminal location when attached to the computer. Computer security itself deals with the ability of the user to input adequate

information to pass a software-oriented sign-on test, established in the computer software, given the terminal had an acceptable physical and logical configuration. Application security deals with the ability of a user who has passed physical, logical and computer tests to pass further software-oriented tests to allow the user to run an application program. Access security is the terminal-to-network transport connection security that requires a terminal location to pass physical and logical tests while the user passes access and application tests before being interconnected to a processing computer or application. This fifth level of security is just evolving as a need in distributed processing networks, where physical and logical screening tests may no longer be relevant. Gandalf and others provide this access security by physically divorcing terminals, workstations and PCs from hosts and building "switched" connections that are "authorized" by software tables. Figure 1 illustrates the traditional connection and Figure 2 illustrates the access network connection concept.

In a traditional sense, physical security is based on where you put the terminal and how you protect against unauthorized access to the site.

Logical security is typified by IBM's hierarchical SNA structure. Terminals can only perform specified functions that are related to their physical and logical attachments parameters that are spelled out in software tools. Such terminals traditionally had to also pass computer and application screening software, assuming every physical/logical relationship was predefined. ASCII/ANSI type computer systems used this same basic philosophy, even though user locations were not physically defined, nor logically addressed within the computer tables. This inability to physically or logically define a user led to a series of additional computer and application screening programs that were intended to thwart security penetration. As with SNA, the internal software-oriented testing assumed the terminal was physically secure at RS-232/RS-422 distances (50 to 5000 feet), on-premise.

The proliferation of physical and logical configurations that now include public network access (dial-in and/or X.25) presupposes that traditional computer and application security software would be an adequate means to insure information protection from an unsecured source. In fact, public

access created the opportunity for the hacker to match wits with the security provisions of the computer and the application software. Although we have no proof, general wisdom implies the hacker always wins, unless a new level of security (access) is added to the network.

A second physical configuration that increases the risk of security breaches are those associated with the development of non-addressed local area networks (LANs). Typical LANs, using carrier sense multiple access with collision detection (CSMA-CD) schemes, presuppose that the attachment to such a network is tantamount to unrestricted use of the computing resources, transmission resources and application software on the LAN. Ethernet is a typical application of this "party line" strategy. The LAN owner and the network manager must assume that basic computer and application security, plus physical location protection on-premise can thwart any unauthorized usage or unwanted peeking at authorized information. Much like the conventional wisdom that implies the hacker wins whenever public access is allowed, conventional wisdom says the unauthorized user wins against computer and software security measures when

users are connected to a party line bus structure.

The physical and logical shortfalls of the ASCII computer and non-addressed LANs coupled with the wide scale introduction of public access connectivity have led to the deployment of switched security systems, such as Gandalf's Private Automatic Computer Exchange Network (PACXNET). As mentioned, these access security units divorce the user and the public network ports from the computer and allow the network manager to create physical and logical address and screening levels for each terminal, user and access port, independent of where the computer application security resides.

This form of switched access security allows the network manager to revert to the very basics of security provisioning. A specific port or terminal can be given software-defined functionality based on all four levels of security (physical, logical, computer and application). Access security simply relates all four levels simultaneously, before allowing the user to be connected to a computer for sign-on. This additional layer of security makes penetration of computer and application software extremely difficult,

while still supporting public network access and CSMA-CD transactions on a CPU-to-CPU basis. In fact, on the most sophisticated security systems, such as PACXNET, the same user from a different terminal may dynamically redefine form, fit and function of the terminal, but only to the extent authorized by the security software of the network controller.

Computer security may still be breached internally, if an employee somehow finds out the access procedure of another. This kind of breach may allow an unauthorized user to gain access not only to the computer but to specific computer files for which she/he may not be authorized. Sophisticated access security systems, such as PACXNET, protect against this eventually by allowing a computer user to dynamically change their own passwords and by allowing multiple user names and passwords from the same terminal, based on the application requested.

Sophisticated Security

Switched network controllers provide a level of sophisticated security by

acting, in effect, as a "doorway" or "guard" for the computers which they serve. The network units perform this security function by being able to restrict terminals to accessing some computer services or databases but not others.

This kind of destination security requires the network unit to query the terminal user to establish which computer service to access. The terminal user simply indicates the service desired. The network control device determines if the terminal is permitted to access the computer that provides the service. If the answer is positive, the network requests entry of a password associated with the application service. After this password is correctly entered, the network unit determines if the terminal and the desired service and the user are authorized to access specific data files stored in the computer. If the network determines that the terminal is not permitted to access this computer or the application, the connection request is terminated immediately. If the total request package is authorized, the network unit builds a path to the computer and application. This path building is commonly called dynamic switching. In some cases,

the network actually signs-on to the computer and the application (i.e., user does not have to know computer or application sign-on).

This sophisticated network control of computer access and application security by terminal location, destination requested, user name and password convention provides a very effective form of access security, without reducing the flexibility and user-friendliness of network operations. But, each user device is only allowed to access computers and applications authorized to the physical, logical and user names specified by the network manager.

For example, assume an accounting department operates a PC, which is also used to access the computer running an organization's payroll system. For security reasons, no other terminal or PC is permitted to access this same computer and application. But, the president's terminal and the president himself are permitted to access all computers, applications and files on the organization's network. These incompatible requirements are typically handled through dynamic reconfigurations that can be built into the access

control database.

When the president is in the sales office, for example, and wants to use a local terminal to access the restricted computer containing the payroll data, he can invoke the dynamic user/terminal reconfiguration capability. This user-enabled reconfiguration allows access security to be dynamically modified for a user by defining specific hierarchical password and user codes that enable the terminal to perform any function authorized by the network manager. This kind of sophistication is actually an advanced form of access security that is only available on a "networked" device.

Advanced Network Security

The most advanced approach to network security and flexibility incorporates this dynamic ability to reconfigure security operation by user name and passwords. This advanced network security level causes the network, rather than, or in addition to, the destination computer, to decide if a particular user is permitted to access specific computers only from his own

terminal or from a terminal group or from all locations. It is based on network coded user names which have been approved by the network manager and passwords which can be changed by users themselves, at any time, but are otherwise "locked-in" by the network.

Under this approach, a user can access computers in a network from any pre-designated physical or logical location. At the request of the network, the user first enters his/her coded user name. After a network check, the user enters his/her password in response to a message. In this way the security configruation allocated to the user can follow the user around an organization. It also provides for billback and security flags at every location.

Based on the user name, password entries and the physical and logical parameters of the access point, the network determines which computers can be accessed by the user from that access location. This eliminates the need to restrict computer access on the basis of which terminal is being used. The user then proceeds to pass through the same basic or first level

of security as before, by identifying the computer service she/he wants to access and entering its associated password. Once connected to the computer, the user also follows the normal access procedure. This variable user/terminal security is an excellent enhancement to all networks. It is particularly valuable for sites using dial-in access arrangements or for multiple locations that share computers through networking.

In practice, the network determines "who" is calling in and implements security accordingly. Additionally, the advanced network security may also require that a requester be cutoff after the calling sequence has been used. The caller is then called back by the network at a specific phone number or terminal address from which a specific user is authorized to operate or which has been pre-arranged by the network manager. This "dial-back" enhancement can also be used to minimize transport expenses by reversing the calling pattern to the central hub where large trunk groups or bulk tariff services apply.

Advanced Security Benefits

The more critical the information, the more advanced the network security needs to be in terms of achieving maximum network flexibility as well as protection against unauthorized computer access.

In PACXNET, for example, maximum user flexibility is achieved by allowing a user to access any computers from any point in the network. This means controlled computer access can be attained via dial-up phone or X.25 connections from anywhere in the country or even the world without fear of security breaches. In addition, users can even be electronically messaged by name at whatever terminal they sign-on to the network.

Total network security is also improved by divorcing user connections from the computer port. This can prevent unauthorized users from getting into the network, much less accessing a computer file. For example, in direct-attach networks, such as Ethernet, SNA and most LANs, an unauthorized user can get into the computer by simply gaining physical access to a terminal. But, under the advanced security levels of PACXNET,

an authorized terminal cannot gain access to the network or to computers unless the user knows both a network manager-defined computer/application name and a user-specified password. And, if extreme security is needed, PACXNET can even allow the user to change passwords and have specific passwords for specific applications. These features are designed to successfully prevent penetration through expansion of the permutations needed to gain access.

Use of this advanced security level also makes it possible to produce an audit trail of network usage showing which users signed on from which terminals, to which computers, for which applications, on what day and for what time period. Another side benefit of advanced security is that user authorizations can be quickly and efficiently rescinded from the network. By making only one entry, an MIS manager can revoke a user name from the centralized control system, immediately eliminating the access capability from all sites. Hence, there is little danger of penetration by disgruntled employees.

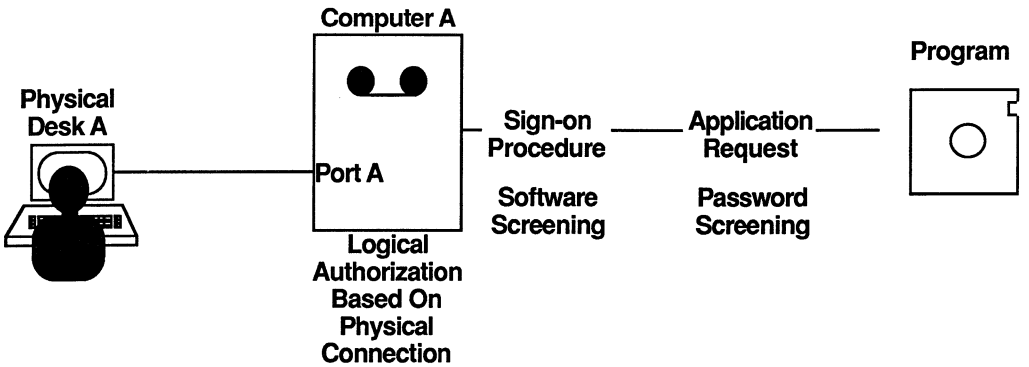
Capability Affected By Security

Information is the lifeblood of an organization. The purpose of networking computers and users is to circulate this lifeblood to all parts of an organization in a timely and flexible manner. Security measures, on the other hand, are used to make sure none of the lifeblood is leaking out. Basic--and even some sophisticated--security measures can sometimes hamper network flexibility while failing to completely close the door against unauthorized access and information loss. The most sophisticated systems, such as Gandalf's PACXNET, divorce terminals, users and computers and build authorized connections that have passed advanced screening tests before allowing a user-to-computer transaction. If the lifeblood is important to the company, it behooves network managers to consider these most advanced levels of security, even if it changes the network communications architecture, expecially if the change also produces more user-friendliness and flexibility.

-end-

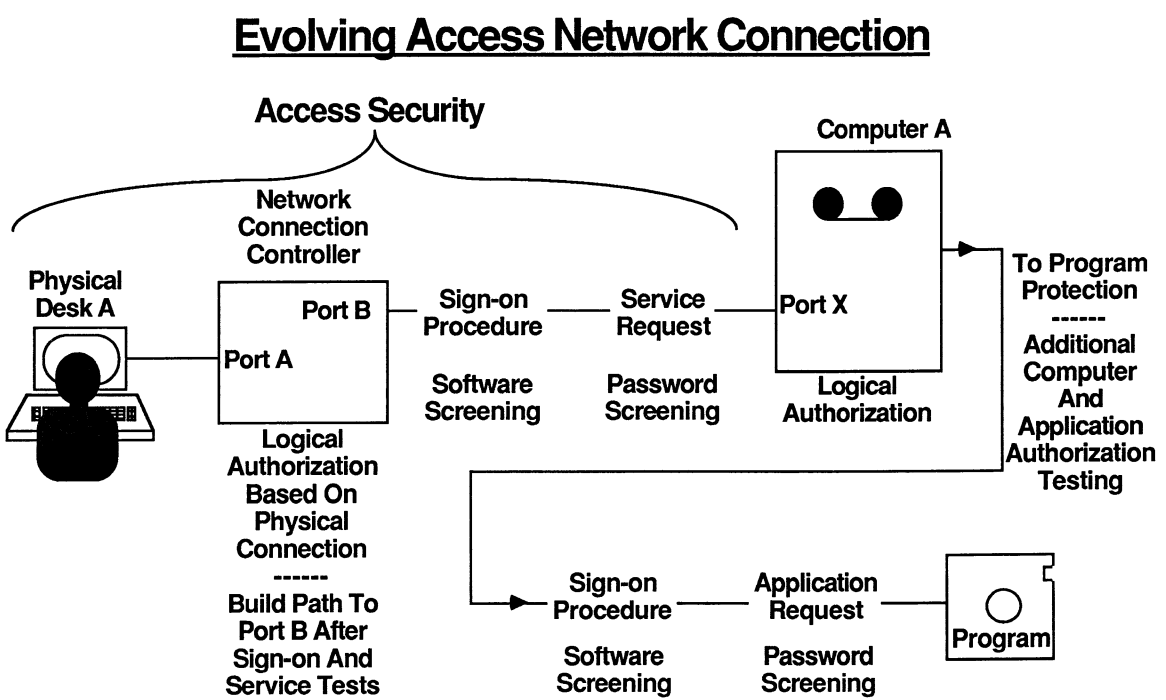
FIGURE 1

Traditional Local Connection



gandalf™

FIGURE 2



gandalf™

ABSTRACT

Improve User Productivity with Menu Handlers

Larry Haftl, Systems Resources

In our profession, "Productivity Improvement" usually refers to creating software faster/cheaper/better. By improving the user/machine interface it is possible to show productivity improvement throughout an entire organization instead of one department. A menu handler utility can be a very powerful, inexpensive tool for improving that interface. This paper will discuss the evolution, current state and uses for menu handlers on the HP 3000. Emphasis will be on increasing end user understanding of and ease of use of system by employing dynamic menus.

THE ROLE OF FOURTH GENERATION LANGUAGES IN THE LIVES OF 3GL PROGRAMMERS

by
Suzanne M. Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito, CA USA 94530-3059

THE ROLE OF FOURTH GENERATION LANGUAGES IN THE LIVES OF 3GL PROGRAMMERS

In my early years of programming, we did everything in assembler language. With assembler language, one nmeumonic equated to one machine instruction and was accompanied by one or more operands such as a register number or memory address. You knew the octal or hexidecimal (base 16-IBM) representation of each instruction and could read a memory dump as easily as you would the Sunday funnies. It was wonderful. It was simple. It was straightforward. The programmer had complete control. When you compiled your source program the machine punched your compiled object out onto a nice little deck of 80-column cards, and if you needed to make just a minor change you could fix the object code card and reinsert it into the deck.

After many years of this secure world, somewhere in the early 70's, the then current boss declared that we were to become a COBOL shop. The reasons, he claimed, were simple:

- COBOL was the way of the future
- COBOL programs could be developed faster than assembler language programs could
- COBOL programswould be far easier to maintain than assembler language programs
- COBOLrequired a much shorter learning curve than assembler language
- Long-range, COBOLprogrammers would be more plentiful and less expensive.

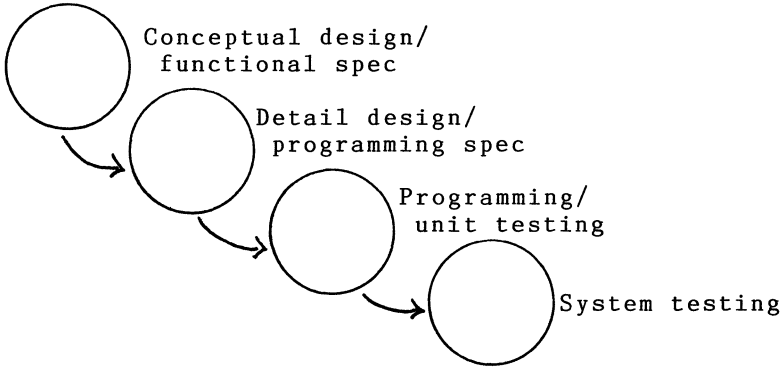
The bottom line - the company would save huge amounts of money on software which was rapidly approaching 33% of the average data processing budget. An additional benefit would be shorter development cycles.

As programmers, our reaction was instantaneous and unanimous -- yuck! COBOL was verbose, slow, a memory hog (we had just upgraded to a 32K memory machine -- virtual memory was still a twinkle in someone's eye), and took away control from the programmer to the point that you would never know what the machine was doing. The only consolation was that we could still get memory dumps, which enabled us to figure out exactly what machine code each COBOL instruction was generating. We were appalled, of course, by its inefficiency.

I had somewhat forgotten this whole experience, now some fifteen years past, until recently. I was discussing with a client the difficulty many 3GL programmers have transitioning into the 4GL world, and he reminded me of our experiences transitioning to 3GL's (though we didn't really use fancy names like 3GL then). It was that discussion which inspired my interest in attempting this paper.

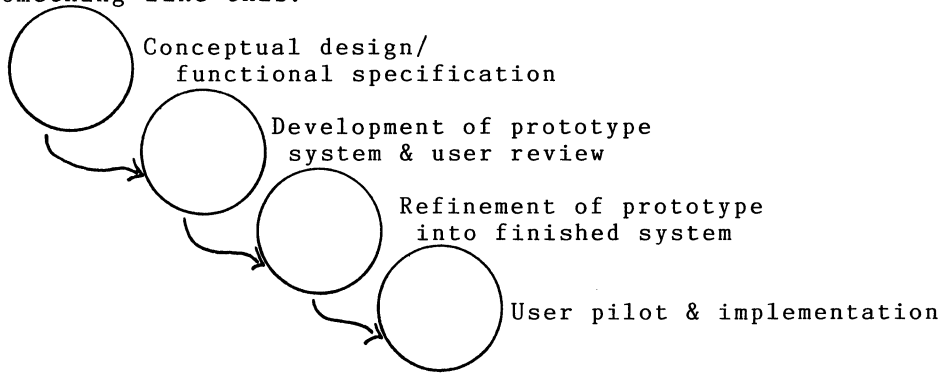
I started working with fourth generation languages about six years ago. I will not say that there have been no frustrations, but I would now refuse to develop a system in a 3GL. I would consider it a waste of my time and the client's money. However, developing with a 4GL is a very different way of approaching application solutions than programming with a 3GL. Therein lies the key -- to be successful with 4GL's we must transition from the role of a programmer to the role of a developer.

In a traditional 3GL development environment, the development cycle proceeds something like this:



When the detail design is accomplished, the system is broken into definable program pieces before programming begins. There is good reason to have solid specifications before coding begins, because making programming changes and repeating test cycles can be extremely time consuming and costly.

In a traditional 4GL development environment (if there exists such a thing), the development cycle proceeds something like this:



There is no detail design phase producing programming specifications. Program pieces are not predefined. For one thing, very few 4GL's, and certainly not the more widely used 4GL's, give you a clearly defined "program" concept. These products are all oriented towards

- a screen
- a report
- a process

The "steps" that a developer must use to accomplish these depends on the 4GL and the task at hand. Keeping this basic difference in mind, let's proceed to the "hurdles" a 3GL programmer faces in becoming a 4GL developer.

Hurdles:

1. Beginning the development cycle without knowing your product (4GL). This is by far the most common problem. There is a new $250,000 project that is behind schedule and critical to someone's career. Management buys a 4GL to "save the day." They expect the MIS staff to wholeheartedly embrace the new product and immediately realize the kind of productivity gains promised by the vendor.

Result: Resentment and/or hatred of the product by most, if not all, the staff. Failure of the project, or at the very least a very bad rep.

Other possible side effects: Trashing the 4GL and returning to a 3GL. Blame placed on 4G1 vendor for misrepresenting product.

Suggestions: Don't use a new 4GL for the first time on a major new development project. Use it initially for enhancements to existing systems or extremely small new systems.

Get proper training. If there are more than a couple of people who will need to know and use the product, get custom training in-house. In the end it will cost you less because your time will be used more effectively.

If you have no choice but to use the product for the first time on major new development, bring in people who really know the 4GL and how to work with it. Let them direct the project and listen to them.

2. Specing the project as you would for 3GL development, then trying to develop it with a 4GL. This is a real killer, and what I otherwise refer to as trying to fit a round peg into a square hole.

Result: Most frequently, the programmer will claim that the 4GL lacks functionality and sophistication enough to do the task and will resort to their 3GL of choice to accomplish the task at hand.

Other possible side effects: Those programs which were written in the 4GL will be incredibly inefficient and may "look" out of place.

Suggestions: Design "into" your 4GL. Each 4GL has its own unique "style." It is critical in developing an attractive, effective, and efficient system that you design for that style. This includes data base design, screen design, report design, user interface, even test plan. In fact, this style is so critical to the way your finished systems will look and act that you should learn as much as you can about it before you even choose a 4GL. This is facilitated by translating the functional spec directly to a prototype, and working the prototype into a finished system.

3. Assuming that you will have the kind of logical control you had with a 3GL. I have found that typically 3GL programmers feel most comfortable with COBOL generators which claim to be 4GLs or 4GLs which are entirely procedural, and are thus as close to being a 3GL as possible. These products will help somewhat with the cost of initial development, but long term productivity gains due to ease of maintainability and shortened development cycles as familiarity with the product increases, will be lost.

Result: What most frequently happens when a 3GL programmer is reluctant to relinquish control to the 4GL is that they take every opportunity to use whatever procedure code is available with the 4GL they have. This usually results in lengthy code to do what could have been done automatically with default or design statements, thus defeating the entire purpose of the 4GL.

Other possible side effects: The programmer feels the 4GL is cumbersome, lacks functionality, and it would have been easier to use a 3GL in the first place. His boss is beginning to agree.

Suggestions: With a 4GL, not only are you not going to have the control you had with a 3GL, but a lot of what you do will be guesswork or trial and error. There will be times, many at first, fewer later, when you could do it much faster in a 3GL than it will take to figure out how to do it with a 4GL. Build yourself a support group. Join your local users group and exchange cards with people who seem eager and willing to share what they've learned. Frequently 3GL "techy's" egos are so big that they are embarrassed to reach out for help with a 4GL. As a co-worker of mine said the other day: "The only stupid question is the one you didn't ask!"

4. Performance is unsatisfactory! If a prospective 4GL user asks me three questions, one of the three will always be: "What about performance?" The answer: "4GL programs are not as fast as 3GL programs." This can be compounded tremendously by the first three hurdles discussed above. Inadequate knowledge about the product, inefficient use of the product, and system designs which "conflict" with the product can all contribute to additional degradation in performance.

Result: The product may only be used where performance is irrelevant, or may not be used at all.

Suggestions: First, review the three hurdles above. To optimize performance you must know your product, get proper training, make enough of a commitment to become experienced with it, be willing to use trial and error, and get support from others who have experience with the product. I have a terrible habit. I build the system first and worry about performance second. However, the advantage is that usually for a period of two to four weeks, I turn my attention totally, or as close to totally as I can, to tuning the system. Many tools are available to help, depending on what the problems are. Supertool, MPEX and Omnidex, as well as many other products, offer tremendous performance enhancements to the average system.

There is, however, another issue where performance is concerned. Unless there is a serious problem with system resources, performance should be in the eyes of the user, not the eyes of the MIS person. Sometimes MIS people feel they must put "performance standards" on systems in order to maintain control, i.e., all screens must have 3-second response time. In fact, if the system has functionality that greatly enhances the user's productivity with 8-second response time, and the user is ecstatic, then it becomes questionable whether there are performance problems.

Conclusion

The 3GL programmer who is transitioning to 4GL developer will need to work on altering their mindset and their entire approach to system development.

The following is a list of ten commandments to help you through the ordeal:

1. I will read a section in my 4GL manual every day.

2. I will try at least one new function or feature of my 4GL every day.

3. I will not say it cannot be done in my 4GL until my 4GL vendor tells me so.

4. I will never resort to procedural statements until I have made absolutely sure I can't do it with design statements.

5. I will go to every meeting of my local users group.

6. If there is no local users group, I will talk to my vendor about helping me form one.

7. Instead of abandoning my 4GL, I will try to rethink problems to take advantage of my 4GL's functionality and features.

8. I will not swear and have a tantrum when I can't find anything in the manuals which relates to my question or problems.

9. I will try at least ten ways to do something new which I don't know how to do and can't find reference to (reduce this by two every six months).

10. If I can't follow these commandments, I will find another job.

Implementing a System using Enhanced Data Search Capabilities

by
Suzanne M. Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito, California
USA 94530-3059

As users gain sophistication and technology pressures us towards a paperless society, improved techniques for complex searches of data base information are becoming a necessity.

Omnidex is a powerful enhancement to the Image/3000 data base management system. Using sophisticated inverted files and binary trees, Omnidex allows rapid retrieval of data records using any Image field identified for Keywording. It also allows retrieval based on any combination of words or values contained within key-worded data fields.

The implementation of a large on-line application system (data sets of 250,000 to 1,500,000) using this product will be discussed. Though development of the application was well underway when the product came to the attention of the project team and the user, the decision was made to retrofit the system to take advantage of Omnidex.

The focus will be on:

- Why the decision was made
- The anticipated and real costs of the decision
- Where Omnidex was used and where it was not and why
- What the data base and application design considerations were
- Discoveries made along the way
- How users perceive the system and the functionality Omnidex provides
- Performance benefits and prices

The Environment

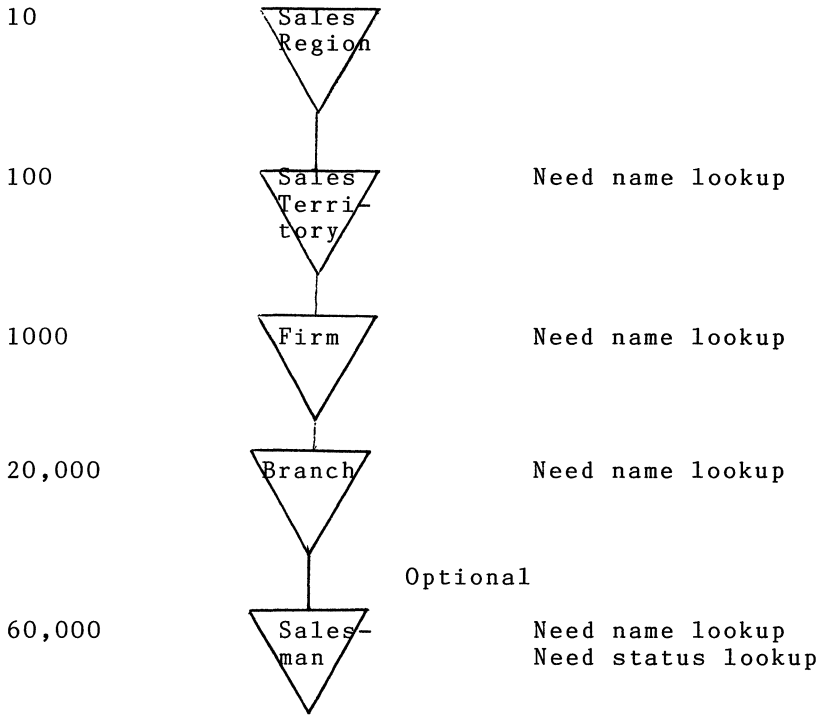
An Investor Services System for a large Real Estate Investment firm.

- Large Data Bases
  - 60,000 Salesmen
  - 500,000 Investors
  - Many data sets over 1,000,000 entries
- Hi volume of on-line inquiries and changes
- Dynamic organization with frequent system changes to support marketing driven business

The Problems

Problem No. 1

Logical Data Base Structure -- multi-level hierarchical relationship of the sales "organization"

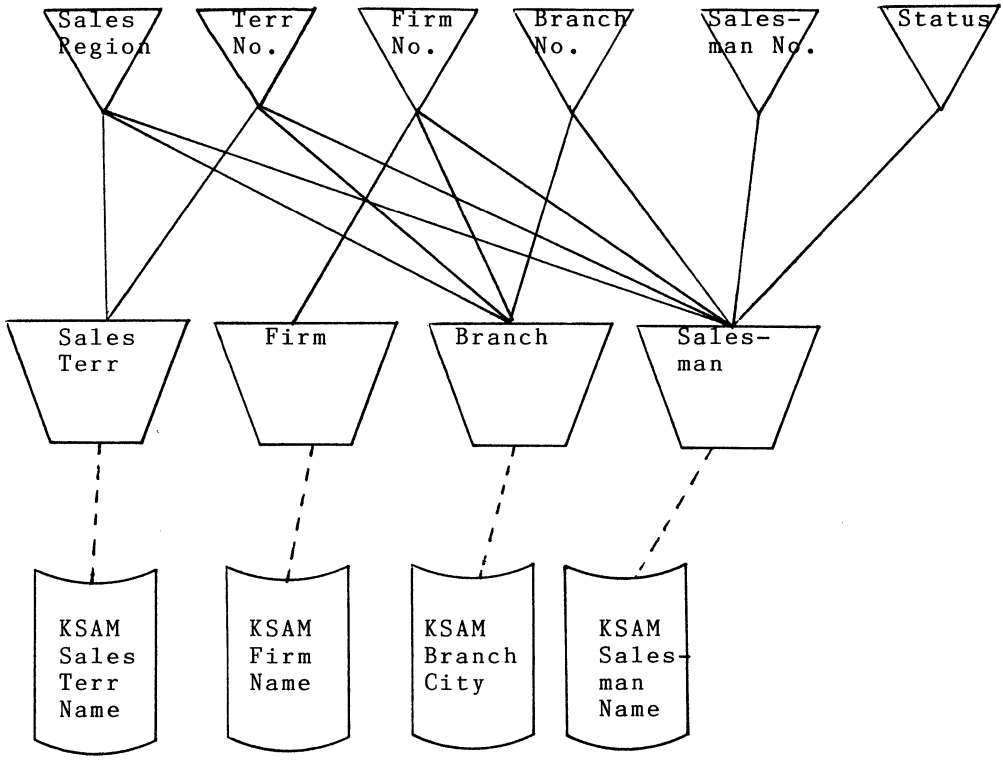


Salesman may be temporarily "unassigned"

The first problem was a multi-level hierarchical data structure representing the sales "organization."

In addition to the need to search for data based on higher levels of the hierarchy (i.e., need to be able to search for branches by region, territory and firm), name lookups were required on most sets.

Image Data Base

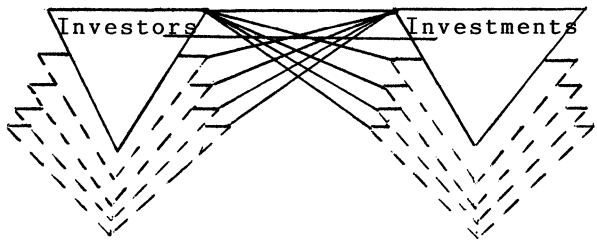


Disadvantages:

- Up to six paths per detail
- Not an "all image" solution
- Generic name (city) search only
- No multi-keyed access

Problem No. 2

Logical Data Base Structure -- Investors & their Investments
versus Investments & their Investors



An investor can own or have a "relationship" with an unlimited number of investments.

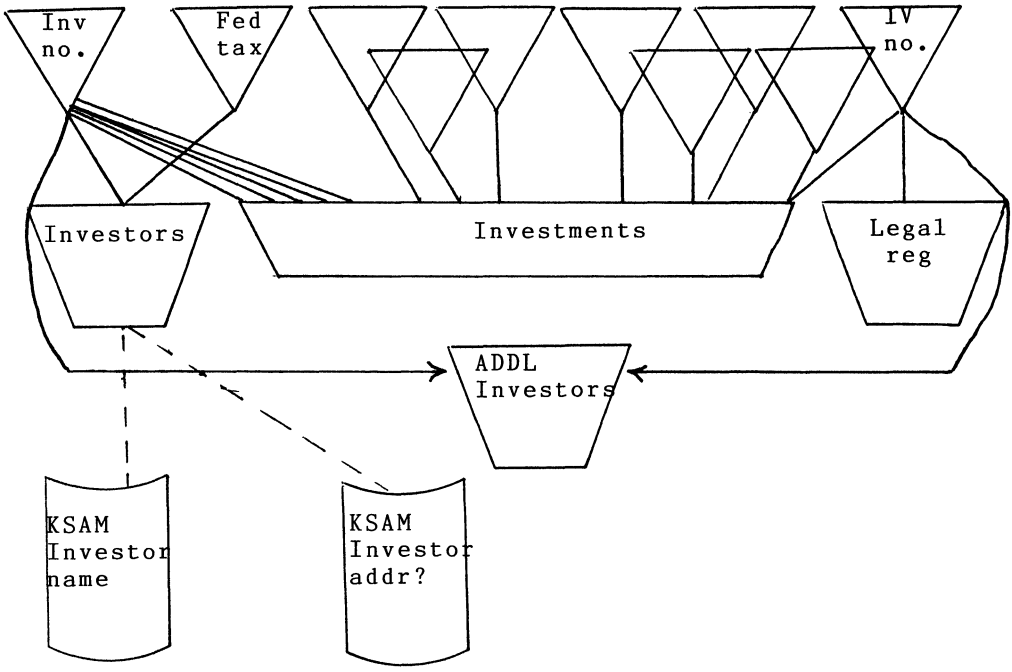
An investment can be owned by or have a "relationship" with an unlimited number of investors.

When an Investor calls, we must be able to look at all investments he/she is related to.

On Investors we must also have a minimum of:
- name lookup
- SS no. lookup
- address lookup

On Investments we must also have a minimum of:
- (Fed tax id no. lookup)
- Input batch/seq no. (from Order Processing) lookup
- Deposit batch no. (from Order Processing) lookup
- (Fund lookup)
- Firm lookup
- Branch lookup
- Salesman lookup
- Institutional Acct no. lookup
- Date of purchase lookup
- (Legal Registration lookup)

Image Data Base



Disadvantages:

- Twelve paths needed to the most important set
- Not an "all image" solution
- Generic name/address search only
- No capability for legal registration search
- Fed tax id and fund chains too long for image, in some cases Investor no. chains too long for image
- Cannot find all investments in any way related to a single investor with one search
- No multi-Keyed access

About six weeks into development, we were given a demo of Omnidex, a then very young product. It was clear twenty minutes into the demo that the enhanced data search capabilities offered by the product met our needs perfectly in several areas:

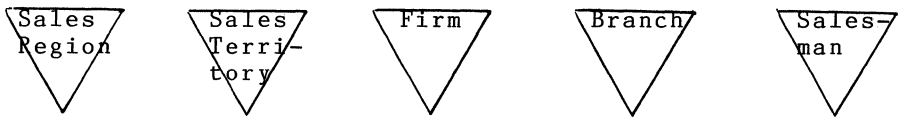
- Works within image
- Provides keyed access without image keys
- No limit to number of "keywords"
- Allows for generic search, Keyword in context search, boolean searching (i.e., and, or, not, etc)
- Keywords may be "grouped"
- "Chains" may be of unlimited length
- Repetitive, non-meaningful words may be excluded from "keyword list" (i.e. Street, and, or, the, etc)
- Can search on more than one Keyword at a time
- Extremely fast searches on large volumes of data

What it means to our design:

Problem No. 1

- Logical masters become physical masters
- Name and address searches are completely flexible

New Data Base Design:



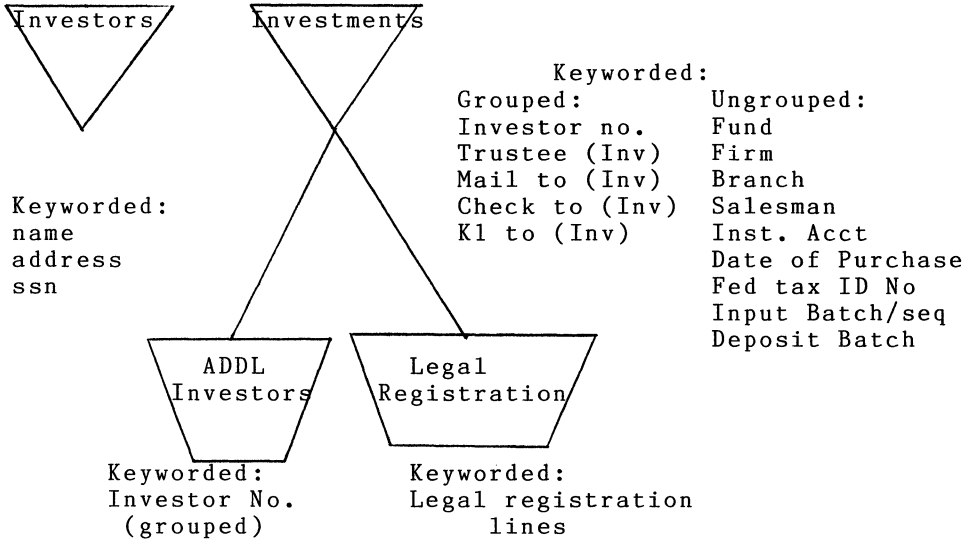
Not Omnidex	Keyworded: Name	Keyworded: Name Address	Keyworded: Region Territory Firm Address	Keyworded: Region Territory Firm Branch Status Name

What it means to our design:

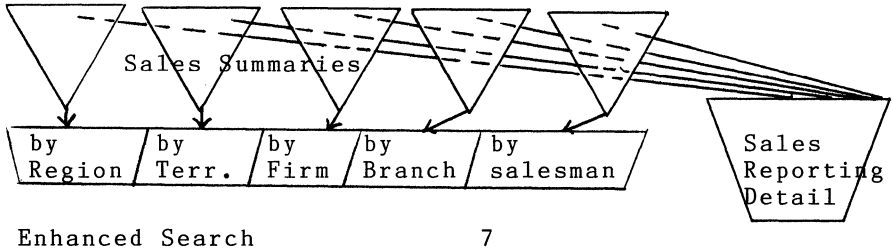
Problem No. 2

- Logical masters become physical masters
- Name and address searches are completely flexible
- Items whose chains were too long for Image can now be Keyworded
- Complete Keyword in context available for legal registration
- Can find the exact investment sought by searching for more than one Keyword at a time i.e., legal reg: Suzanne Harmon
  Purchase date: 041586
- Can find all investments "related" to an investor

New Data Base Design:

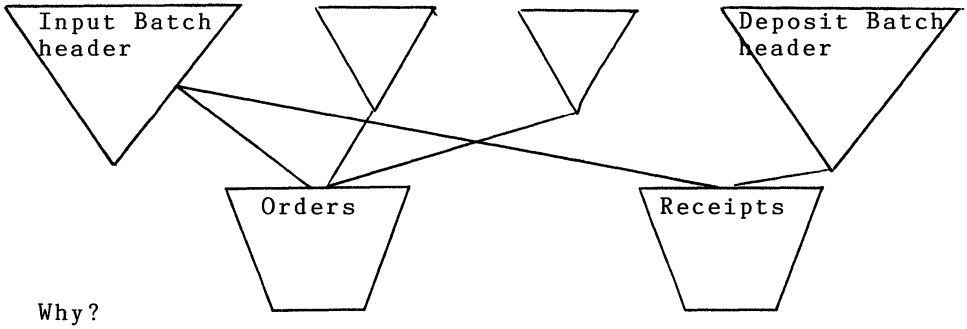


What we did not Omnidex:
Sales History Data Base



Why?
- Accessed on-line infrequently, and then only by Key
- Used primarily for a plethora of batch produced sales reports

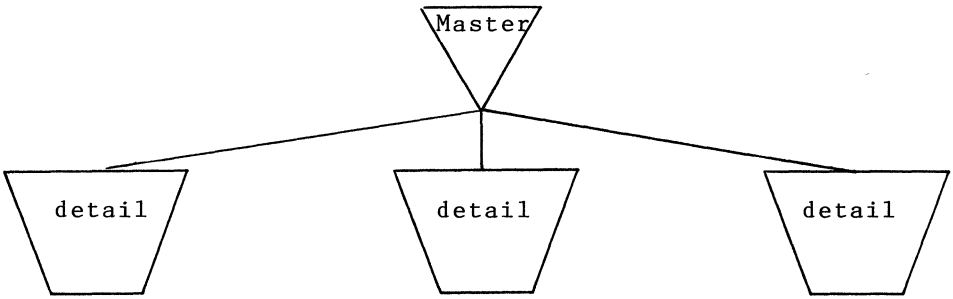
Order Processing Data Base



Why?
- Heavy input, very little inquiry, and then almost always by Batch number
- Orders rarely of interest for more than 24 hours

Design Consideration:

An Omnidex "domain"



- Make "logical masters" image masters. A logical master usually represents a person, place or thing that has a "life" of its own, i.e., Customers, Orders, Parts, Vendors Employees, etc.

- Put occurrence, transaction and optional information in image details, i.e., order lines, comments, payments, job assignments

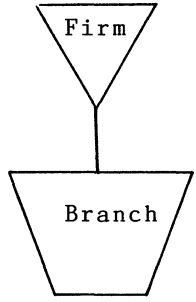
- Keyword all fields which will enhance on-line inquiry or provide "extracts" for reporting on subsets of less than 50% of the data base

- Don't expect to get it perfect the first time around; changes are relatively easy

- Exclude "words" which are obviously unnecessary

Discoveries along the way:

1. Could not do record-specific finds on keyworded details i.e., alternative design



A search on Keyworded item in Branch, i.e., City = San Francisco, would return all the firm no.'s of firms with qualifying Branches, would have to read down the chain to find the specific branch.
Solution: Stick to design considerations discussed above.
Note: Record-specific finds have now been added with the newest release of the product.

2. Excluded Keywords do make a difference.
For example, during conversion had 250,000 records with Fed-tax-id = 0's. This, in effect, was a 250,000 record chain. Changing and deleting records became relatively high overhead. In this case, 0's should have been excluded.

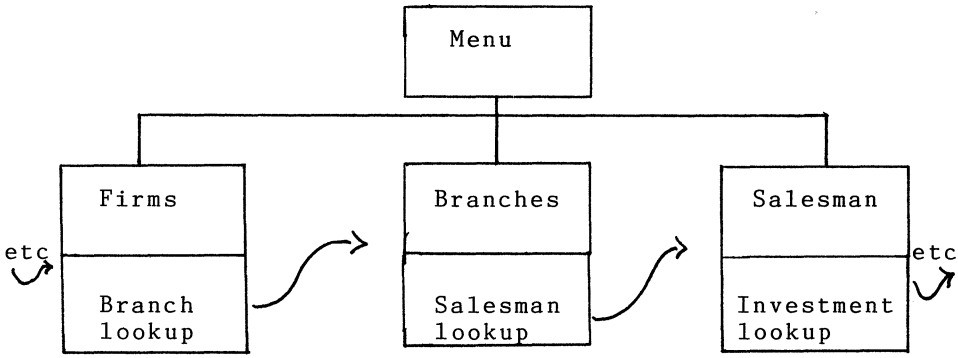
3. Single Set Reloads
   Each Omnidex domain creates three additional Image data sets at Omnidex install time, one master and two details. The two details should be reloaded frequently, depending on volatility of data, for maximum performance. This is a very easy task using DBMGR by DISC which has a Single Set Reload facility, or any number of other utilities available.

4. Initially, all Omnidex masters had to have J2 Key. J2 Keys, in fact, are very efficient because the Key value is effectively the record location. This is fine, unless the highest value of your J2 Keys minus the lowest value ever exceeds the capacity of the data set.

5. Initially, you could not do a find where more than 13,000 records qualified. Since many of our "chains" had more than 13,000 entries, our reporting capabilities were significantly impacted, and we had to switch to Supertool extracts in some situations. The find is now unlimited except for generic finds, ranges, and "multifinds".

6. Because of the size of our data base, re-indexing (i.e., due to adding or deleting Keyworded entries, etc.) originally would have taken a week. DISC responded immediately by improving the Buildfast utility so it now takes 12 hours.

7. Stack-Size Considerations
   Because the system is designed for user-friendliness, i.e...



   And is written in a 4GL, we experienced stack overflow problems along the way. DISC responded by making certain stack efficiency enhancements, and helped us identify

changes to the way we were doing things which cut stack somewhat! Stack still remained a problem in certain "multi-depth" screen situations.

Solution: Used certain "user-transparent" tricks to reduce the depth of screens.

8. Learning Curve
Took one week to convert the Firm/Branch/Salesman system from traditional Image to Omnidex.

Performance and Overhead

OMNIDEX		IMAGE
One Key find	=	One Key find
Multi Key find	=	One Key find then read serially to find add'l keys
Generic, Keyword in Context, ranges, etc		No Equivalent
Put new record- must set-up Keywording	=	Put new record- must set-up paths
Change record (no Keys affected)-simple update	=	Change record (no Keys affected)-simple update
Change record (Keys affected)- update with Keywording changes	=	Change record (Keys affected)- delete record, add record, with all path adjustments done twice
Omnidex data sets added to Image DB	=	Auto Masters + path overhead/record

User Satisfaction and General Comments

- Users love the system
  - Fast inquiry without having to know numbers and codes
  - Much more service oriented for customer
  - Avoids duplication of information
  - Alleviates need for hardcopy listings
  - Expanded system capabilities

- Easy to Support and Use
  - DISC an outstanding "support" vendor
  - New releases are frequent and address installed base needs
  - Easy to learn and use
  - Interfaces with almost all 3GL's and 4GL's
  - Superior documentation

ABSTRACT

4GL - The Controversy Rages On

Karen Heater, Infocentre Corporation

The HP 3000 community is buzzing about application development and fourth generation Languages. This paper will review the definition and direction of fourth generation Software Tools relative to their third and fifth generation counterparts, their associated benefits and the pitfalls which must be avoided in order to ensure successful implementation.

The definition of 4GL's requires a discussion of industry trends and emerging directions. Once having defined and categorized the technology and the variety of product types which fall into this category the paper will discuss that the impact implementation of a fourth generatioin strategy will have on both the existing DP personnel and the users.

Without proper evaluation criteria and expectation settingthe implementation of the 4GL can become unnecessarily time-consuming and costly.

The goal of the paper is to assist the HP 3000 community in assessing, selecting and successfully implementing the fourth generation software. As such, each of the stages from evaluation through to implementation will be addressed through a discussion of the appropriate goals, criteria, expectations and approaches which will take the organization through this cycle smoothly and successfully.

ABSTRACT

4GL and The Changing Role of the Programmer

Karen Heater, Infocentre Corporation

Application development using fourth generation Programming Languages is now a reality. The existing data processing professionals are competent, educated and comfortable in the use of traditional third generation programming tools and methodologies.

With fourth generation Software, the task of implementing application software systems becomes one of telling the HP 3000 what to do but, not necessarily, how to do it. The job entails the intricate involvement of the various end users.

This paper will focus on the changing role and expectations of the programmer in HP 3000 data processing environment as a result of the implementation of fourth and higher generation programming tools. It will address the changes in the required skill set and the considerations necessary to ensure the smooth transition to programming of the future for today's existing data processing professionals.

ABSTRACT

Networking the Mini and the Miro — Distributed Application Processing and How To Use It

Karen Heater, Infocentre Corporation

As fourth generation software and data communications technology becomes more prevalent throughout the HP 3000 community, the opportunities available for networking the mini with our micros grows significantly.

With many organizations reaching and exceeding the computing capacity of their HP 3000, the concept of redistributing the load and moving some of the application processing to the micro computer becomes very attractive.

Distributed processing is a reality today, but for many shops it represents a new frontier that should be approached with caution, in a premeditated way. This paper will take a close look at Distributed Application Processing, involving the networking of the HP 3000 with micro computers. We will first introduce some of the concepts at work, then describe the various data communication topoligies that can be implemented. Having set the foundation for the discussion, we can move on to application design possibilities, investigating how new applications might be designed in order to capitalize on the system resources made available through the networked configuration. Along the way, we will be providing some guidelines for effective use of this "Distributed Application Processing" concept based on the capabilities, strengths, and weaknesses of the various system components (both hardware and software) within the network.

Never Take the Default

by **Jeff Hecker**

Apogee
4632 W. Frankfort Drive
Rockville, MD 20853

Ask some of the many users of HP3000 computer systems why they prefer it to the competition, or to their old system. The typical response will be because it's "friendly" or because it's "easy to use". The term "user friendly" generally refers to the interactive nature of the operating system. The alternative to the interactive system is the "batch" system, which is not very easy to use at all. Un-ambigious, non-cryptic commands and clear, understandable error messages are also "user friendly". But "user friendly" also refers to default values for optional parameters.

To appreciate just how much default values contribute to ease of use, just consider the MPE `:FILE` command. Imagine the effort required to merely list all of the parameters, not to mention choosing appropriate values for each. Now consider some of the other MPE commands with nearly as many options -- `:BUILD`, `:RUN`, `:HELLO`, `:JOB`. Without the defaults for these commands, how friendly would such a system be?

To new users of HP3000 computer systems, (programmers, managers, and ordinary folk alike) the usefulness of default values becomes immediately obvious. Often, defaults are brought to light on day one, when an HP system engineer installs the computer. While explaining a command, a frequently used phrase is "Oh ... don't worry about that. The system will take the default."

While this is clearly one of the advantages of having purchased an HP3000, it sets an undesirable precedent. Most people are lazy. The "don't worry about it" philosophy encourages laziness. Or at least does nothing to discourage it. Explaining options to new users of the

system is confusing. But once a user is familiar with the system, only laziness prevents more sophisticated and more efficient use of MPE.

Quite often it will turn out that the system default on a given parameter is in fact the best choice. But equally as often, choosing a more appropriate value can improve system performance across the board. From system configuration and system resources, to CPU efficiency and user response time.

There are many kinds of defaults. The obvious ones are the default values in the various system commands already mentioned. But there are others. There are "default" ways of doing things. There are "default" algorithms used by programmers. There are "default" system configurations. There are "default" names given to things. Obviously, some of these are more "style" or "procedure" than default, but the idea is the same.

Throughout the remainder of this paper, examples are given to illustrate a point. Sometimes the example is frivolous or contrived, but the idea is to consider the options and to make a conscious, intelligent decision. Not to just take the default because it's the default.

System Security

The prevention of unauthorized access to computer systems is a very popular topic of discussion these days. And not only in the HP3000 community. It's not that there has been a rash of reported break-ins at HP3000 installations, but there has been so much publicity recently about this aspect of system security. There are other, less spectacular, but more realistic aspects of system security, such as -- data integrity, system backups and backup tape management.

Ask an HP3000 system manager what the biggest threat to the system's security is, and he might say "unauthorized access via dial-up telephone lines". He would be wrong. Ask him to list all of the causes of lost data from the system and he will reply with some of these:

1) MPE System Failure
2) Hardware failure
3) Accidental purge of a file
4) operator :RESTORED files instead of :STOREing them
5) Tape I/O errors on backup or DBUNLOAD tapes

Actually, very few HP3000 computer systems have ever been broken into as the result of random microcomputer searching for modem answer back tones. Even then, most attempted break-ins are from ex- or current system users. After all, they already know the phone numbers. Certainly no one is not going to try breaking-in from the terminal in their office. The console error messages would give them away. The anonymity of the public switched telephone network is much more assured.

Even so, the proper system configuration and selection of configuration options can help prevent many of the weak links. Starting at the top, every system manager dreads coming in to work in the morning and seeing this on the system console:

```
      ...
         ...
            ..
02:10/#S289/102/INVALID PASS FOR "MANAGER SYS,PUB" ON LDEV #42
02:10/#S289/102/INVALID PASS FOR "MANAGER.SYS,PUB" ON LDEV #42
02:11/#S289/102/INVALID PASS FOR "MANAGER.SYS,PUB" ON LDEV #42
02:11/#S290/119/INVALID PASS FOR "MANAGER.SYS,PUB" ON LDEV #42
02:11/#S290/119/INVALID PASS FOR "MANAGER SYS,PUB" ON LDEV #42
02:11/#S290/119/INVALID PASS FOR "MANAGER SYS,PUB" ON LDEV #42
02:11/#S291/105/INVALID PASS FOR "MANAGER.SYS,PUB" ON LDEV #42
02:11/#S291/105/LOGON FOR: MANAGER.SYS,PUB ON LDEV #42
      ...
         ...
            ...
05:22/#S291/105/LOGOFF ON LDEV #42
```

This is what most people think of when system security is mentioned. But this is easily prevented. Nearly every user in the HP3000 community knows about MANAGER.SYS. Certainly every user who would attempt to break into an HP3000 computer system knows who MANAGER.SYS is. A potential evil-doer does not even have to guess at a logon ID.

How is this easily prevented? Ever changing cryptic passwords?? Sophisticated logon verification programs??? Around the clock system operators???? Dial back modems?????

No. The most effective solution is the most obvious one. It's a shame that you can't just remove MANAGER.SYS from the system. MPE itself "runs" as MANAGER.SYS. However, logon access can be taken away from MANAGER.SYS by removing IA (Interactive Access) capability.[i] In the same vein, remove MGR.SYS and OPERATOR.SYS as well. These user IDs are merely the default system manager and system operator logon IDs. There is no reason to necessarily leave them on the system. Why give someone a head start on breaking into the system just for knowing a little bit about HP3000s? Instead of MANAGER.SYS, use something like "I1006S5G.SYS". Even system users who see the entry in a :SHOWJOB listing will not know that it is the system manager or system operator.

There is no requirement that the system operator even be logged on in the SYS account. Just because MPE does the operator a favor by automatically submitting :HELLO OPERATOR.SYS, there is no reason to leave it that way. Not every HP3000 in the world needs identical logon IDs. This only gives a break-in attempt a head start.

At the same time, and for the same reason, eliminate FIELD SUPPORT, MGR.HPOFFICE, MGR.HPPL85 and MGR.TELESUP. These accounts and user logon IDs are provided by Hewlett-Packard with the same capabilities, passwords and lockwords on every HP3000 in the world.

* * *

In general, when creating new users, there is another factor to consider when selecting options to the :NEWUSER command. The ;HOME= option assigns a default file group at logon

time. For some users it might be desirable to force the user to enter a file group at logon time. Consider MANAGER.SYS or MANAGER ACCOUNTING or some other user with special capabilities. To keep random, unauthorized access attempts from succeeding, force the group name to be supplied.

When no home group is assigned to a user, and the group has a password, MPE will require the group password to be entered also. Even if the user is an account or system manager. When assigning new groups with the :NEWGROUP command, the default for ;PASS= is no password The theoretical microcomputer random letter generator would take thousands of times longer when required to supply the group name and a group password. If passwords are the only obstacles to access a system and it's data, then be sure to actually use the capability provided by MPE.

* * *

One of the dilemmas faced by the designers of "user friendly" systems is the conflict between user friendliness, and system security.[ii] How much help should the system offer when an error is encountered? When MPE was designed, there was not nearly the concern in the DP industry over security as there is today. The MPE designers chose for the user.

Suppose a sequential number dialing microcomputer finds an HP3000 dial-up line. When some random text is entered, MPE kindly responds with:

```
EXPECTED HELLO, :JOB, :DATA OR (CMD) AS LOGON.  (CIERR 1402)
```

System managers who fear random break-in attempts, should realize that MPE actually leads the way into their system. Explicit, clear, easy to understand error messages are given right along the way. After "**:HELLO**" is correctly entered, MPE then proclaims that it would like a user name, then an account name, then a password, etc ... etc.

Fortunately for concerned system managers, this is merely the MPE default. All of the warnings and error messages are stored in a disk resident file. Instead of the helpful message above, a security conscious system manager can change it to something of little use to a break-in attempt. Something which still conveys the general idea that an error has occurred. Something like:

```
NOPE.
WRONG.
SORRY.
```

There are quite a few different helpful messages of this nature which could be replaced. Messages explaining everything from "NAMES MUST BEGIN WITH AN ALPHABETIC CHARACTER" to "NAME GREATER THAN EIGHT CHARACTERS LONG" and "UNEXPECTED SPECIAL CHARACTER". The messages are stored in the file "CATALOG.PUB.SYS". The MPE System Manager/ System Supervisor reference manual contains information on how to modify or replace any message in the file.

* * *

More likely, and therefore more threatening than a break-in, is accidental corruption of data. It can happen so easily by running the wrong program, submitting the wrong job, or purging the wrong file. A break-in attempt succeeds once in a blue moon. Purging the wrong file or data item happens every day of the week. When creating new users and new groups, there are ways to help prevent these sorts of accident. By assigning combinations of ;CAP= with :NEWUSER and ;ACCESS= with :NEWACCT and :NEWGROUP commands, some accidents can be prevented. The simplest case is the accidental :PURGEing of a data file or program. By having users logon into one group, and having important files in another group, the ;ACCESS= parameters can be used to deny SAVE access to non-group users. Similarly, by granting READ and EXCECUTE access, there should never be a need to :RELEASE anything. Another default of the :NEWUSER command, SF (Save Files), can be removed from users who have no need to create or delete disk files.

Similarly, there are often programs and job files which are run daily or weekly or monthly or whenever, which update data bases or running reports. Programs which make these changes are most often run from jobs. Accidentally running one of these programs out of sequence can cause duplicated data in a data base or other corruption. Programs of this nature can be kept from being run accidentally by taking away IA (Interactive Access) from the program file. They can still be run from jobs by leaving BA (Batch Access) intact.

IA and BA combinations are assigned with the :NEWUSER :NEWGROUP and :PREP commands. Programs which are only to be run from a job, only need BA. Users who do not stream jobs only need IA.

* * *

Another all too frequent cause of lost data is re-recording over backup tapes, or :RESTOREing a group of files instead of :STOREing them. Both of these accidents can be prevented by using labeled magnetic tapes instead of the default unlabeled tapes, for offline storage. Labeled tapes incorporate an expiration date in the header records. When mounting a labeled tape, MPE attempts to read the header records assuming that it is a labeled tape. If an invalid header is read, the tape is mounted as an unlabeled volume. If it is a labeled tape, and the expiration date hasn't passed, then the system will write protect the tape volume. When reading or writing a labeled tape, the file name given in the :FILE command must match the volume name on the tape. This makes mistaking one tape for another much more unlikely.

There has been much ado about MPE's ability to correctly read and write labeled magnetic tapes (either IBM or ANSI standard). Years ago, this was true enough. But recently, HP has finally been able to handle labeled tapes in latter day revisions of MPE.

System Resources

Probably the single most tangible system resource is free disk space. Utilities exist to monitor the amount of free disk space, re-block and copy disk files. Surprisingly, given the obviousness of disk space usage, optimization is rarely attempted, and even then, never on a

large scale. HP3000 system managers seem to perceive disk space as an entity beyond their control. They simply wait until there is no more free space left, and then call their HP sales representative to buy more. While there may seem to be no alternative, and certainly no self-respecting salesman will turn away a customer, a little bit of knowledge goes a long way in saving disk space.

Every HP3000 system has hundreds, or even thousands of text files which are taking up significantly more disk space than necessary. This is due to the "card image" nature of the MPE file system. An 80 column text file with only a few words per line, or even blank lines, still contains all 80 characters. The trailing spaces are really there. If the file is a numbered EDIT/3000 file then the line numbers add another 10% to the disk space used.

This is merely the MPE default. Fixed length records are only required if the data file is to be randomly accessed (IMAGE/3000 data base files or editor "K" files for example). If sequential access is sufficient, then variable length record text files take up only as much disk space as necessary. EDIT/3000 (:EDITOR) provides an option (SET VARIABLE) which will keep text files in variable record length format. Variable length record files are compatible with most MPE subsystems, most third party software, and with the language compilers. One notable exception is TDP/3000; it cannot handle variable record length files.

There is nothing magical, mysterious, or deficient about variable record length files, but some people and some installations refuse to acknowledge their utility or even their existence. This is probably a hangover from the card reader days when every input card had precisely eighty data characters. No more. No less. The MPE manuals still refer to the first line in a job file as the "job card".

Even so, for people who absolutely, positively must have fixed length records, there is no need to subsist soley on editor 80 byte numbered files. Both EDIT/3000 and TDP/3000 have SET LENGTH commands. these commands set the record width of the text file. There is no need for eighty column files for thirty column data. Imagine a mailing address file. Imagine hundreds or thousands of names. Imagine similar files all over the system, in dozens of different accounts. Now imagine all of the other text files which have only have a few words on each line. All of these files can be kept using much less disk space than they use by default. Some less than half as much.

* * *

Record length and record format are merely two of the many options to the FOPEN system intrinsic. (and similarly the :BUILD and :FILE MPE commands) The second most effective disk space saving option is the blocking factor. Unfortunately, blocking factor is probably perceived as the most confusing and complicated option of all, and therefore most often left to default. Depending on the file size, type, and record size, judicious selection of blocking factor can save almost 50% of the disk space used by a file. (Fortunately, the MPE default blocking factor algorithm will never waste over 50% of a file's disk space)

Understanding blocking factors and being able to choose an appropriate value requires a slight knowledge of the MPE physical I/O system. Data is stored in 128 word entities called sectors. A sector is the only physical unit of storage. All sectors are alike and precisely the

same size. The record size (the "logical" record size) of the file is superimposed onto these sectors by the MPE file system. If a file's logical record size is half the size of a sector, then the file system can fit two records into a single sector. This is a blocking factor of two. Even smaller records can have even higher blocking factors. The highest possible blocking factor is 128. (a one word record is the smallest possible record, and 128 will fit into one sector)

A different default mechanism is used when the logical record size of the file is larger than a sector. In this case, the default blocking factor is always one, and as many physical sectors as needed are used for the block. These disk sectors are stored in contiguous locations on the disk device. If a record is 129 words wide, it will use two full sectors of disk space by default (admittedly a contrived example, but it is the MPE worst case -- 48.6% wasted disk space). It is possible, unfortunately, to choose a blocking factor which is significantly worse than the MPE default. Consider a one word wide file. The MPE default blocking factor would be 128, which is the best possible. If the blocking factor is anything else, then disk space is wasted (The absolute worst case disk file, which cannot be created by default is a one byte wide file with a blocking factor of one, the human's worst case -- 99.6% wasted disk space). IMAGE/3000 attempts to improve the blocking efficiency for data base files, but manual selection can still improve disk space utilization.

Improving disk utilization using the blocking factor involves forcing MPE to put more logical records into each block. This reduces (and most often can eliminate entirely) the wasted disk space used by a file. Take a file with 192 word records. Using the default, two disk sectors will be required. This will leave 25% ((256-192)/256) of the disk space wasted. By forcing MPE to use a blocking factor of two, enough disk sectors are used to hold 384 words, or three disk sectors, with no wasted space.

The same principal can be used for large record width files and small record width files alike. A blocking factor value can be found which reduces or eliminates the wasted disk space used by fixed record length files. A forty word wide file will have a default blocking factor of three (128/40 = 3). This leaves 8 words per sector unused, 6.25% wasted. A better choice is 16 records per block using five sectors and no wasted space. Blocking factors are are specified in the `FOPEN` system intrinsic call or in the `;REC=` parameter of the `:BUILD` and `:FILE` system commands. Blocking factors for IMAGE/3000 data base files are set with the `$CONTROL BLOCK=` control statement.

Unfortunately there is no such thing as a free lunch. When increasing block sizes to very large values, I/O system performance must be considered as well. More on performance later.

* * *

Hewlett-Packard provides utility programs with MPE to display the current state of the disk free space. These are frequently run by system managers and by ordinary users as well. This is particularly true when a program is to be run which creates a large output file or uses a large temporary scratch file. After running FREE2 or similar program and verifying that there is plenty of free disk space, it is still possible to encounter a File System Error number 46 -- Out of Disk Space. This error message is somewhat misleading, since there clearly is

plenty of free disk space. The root cause of this error message is the MPE file system's partitioning of the disk space known as "extents".

Extents are portions of the file which reside on contiguous sectors of the disk drive. The entire file does not have to fit into a single area on the disk. The entire file does not even have to fit on a single disk drive. Files (and parts of files) are dynamically assigned disk space by MPE when they are created. There are two philosophies of disk space allocation. One is exemplified by IBM DOS. DOS allocates the entire disk file on a single drive and on contiguous sectors of the drive. If there isn't a single free area big enough to hold the entire file, then the operation fails. The second method, exemplified by the UNIX operating system and any number of micro operating systems, is to only allocate file space when it is actually needed, one sector at a time. With this method, all free spaces are identical size sectors. A file operation will not fail until there is absolutely no more free space left on the disk drive.

The advantage of the first method is performance. Once a file is located in the disk directory, there is very little head movement since the entire file is stored in one place. The disadvantage is that disk space fragmentation will eventually produce smaller and smaller free spaces until there is effectively no more disk free space. (actually there are no more free disk spaces big enough)

The advantage of the second method is that there is never a problem allocating disk space until it is all gone. The disadvantage is in the overhead of keeping track of all of the little pieces of the file. Directory entries take up a significant portion of the file's disk space. Also performance suffers because of the additional head movement from one part of the file to another.

MPE uses extents as a compromise between the two methods. Extents are the individual pieces of a disk file. Each extent of a file is allocated on contiguous sectors of disk, but the extents themselves are not stored on contiguous sectors. MPE limits the number of extents to 32. This limits the amount of directory overhead required to store the pointers to each portion of the file. When necessary however, the number of extents can be forced to one, causing MPE to store the entire file on contiguous sectors of disk. The MPE memory management system requires program files to be stored this way, for fast access during segment swapping.

The default for the MPE `FOPEN` system intrinsic (and the `:BUILD` and `:FILE` system commands) is eight extents. Extents are allocated only when actually needed. When allocating disk space for a file with eight extents, MPE will need one eighth of the total file size for each extent. Since each extent is stored on contiguous sectors of disk, if there is not a single area large enough to store it, the operation fails with a File System error 46, out of disk space. The same disk file created with sixteen or thirty-two extents would require correspondingly smaller and smaller contiguous disk spaces for each extent.

Since the HP3000 is a multi-user, multi-tasking computer system, it is likely that more than one person or program is running at any given moment. Each of these users is accessing disk files, either explicitly or implicitly via the memory manager swapping segments around. With these considerations, it is unlikely that having a fewer number of extents actually improves performance since other users are contending for the same disk heads. Therefore

increasing the number of extents will not significantly degrade performance and will lessen the likelihood of an "Out of disk space" error, by reducing the extent size.

* * *

One frequent cause of system hangs and system failures is SPOOLer shutdowns. MPE uses the same file system for users' files and for SPOOLed files destined for the line printer. Hence MPE has the same problems creating files. If there is not enough free disk space to allocate another extent of a spool file, then the system can slowly grind to a halt. MPE creates all SPOOLed files alike, regardless of the parameters in the FOPEN call. The SPOOL file size and extent size are fixed at system initialization, either when booting, or performing a :SYSDUMP. Unfortunately, if a given system is used to run extremely long reports (in the thousands of pages), then SPOOL file size must be set accordingly high. But if only short reports are run, or program listings and the like, then SPOOL file extents can be set relatively small. Remember, every job, every running report, and every open list file to the line printer, are creating SPOOL files whose extents may be far larger than necessary. If the large free areas of disk space seem to be swallowed up very quickly after system initialization, look to reducing the SPOOL file extent size to save some disk space.

* * *

The second most recognizable system resource on most HP3000 systems is main memory. More main memory is one of those oft heard cure-alls for whatever ails a system. Sometimes this is true enough. But it doesn't mean that more memory need be purchased. Again, buying more memory can be considered the default solution. By judicious allocation of memory in programs as they are written, memory space (as well as MPE's memory management performance) can be improved. All HP3000 computer systems have a limited amount of memory (although some are more limited than others). The MPE operating system compensates for this hardware limitation, to an extent, with Virtual Memory (VM) for program read-only code. VM is the process of placing data not currently being used into secondary storage (usually disk) so that some other data can be moved into main memory. The MPE memory manager works very well, and insight into how it works can improve program and system performance even further.

When a program references data which is not currently in main memory (either a program code segment, or an extra data segment), MPE's memory manager is invoked to retrieve the data form disk. The first task is to make room in main memory for the data segment being fetched. There are two distinct types of data segments. Read-only, program code segments, and read-write data segments (including process stacks, extra data segments, file system buffers and disk cache buffers). Nearly every time a segment is fetched, it must take the place of some other segment(s).

MPE will always attempt to overlay program code segments first. There is no need to write a code segment to disk because it is read-only. There is already a copy on disk. If enough space can be found in main memory for the new segment, then it is loaded from disk.

If there is not enough main memory space available by overlaying program code segments, then read-write data segments must be written to disk. This effectively doubles the memory

manager overhead. First, the old data segment(s) must be copied out to disk. Second, the new segment is loaded into memory.

There are several ways to take advantage of the MPE swapping method. By keeping segments small, fewer loaded segments are overlaid or swapped out. With MPE-V, disk caching uses all of the main memory not otherwise allocated to data segments. Since these are read-write segments, they must be posted to their respective disk drives before being overlaid.

Keep isolated code modules in isolated segments. If the program initialization code is in the main segment, then it must be loaded into memory every time the segment gets swapped. Such code should reside in its own segment. Once it executes and is swapped out, it will never be swapped in again, relieving the memory manager of the additional work.

* * *

Every open file used by a program is allocated file system buffers by MPE. These buffers are large enough areas in main memory to hold one block of records. There are two ways to eliminate excess buffers when they are really not needed. The first is the most obvious. Close disk and tape files when they are no longer accessed. All too often, programmers get lazy and assume that MPE will clean up after a program finishes. This is usually true, but these files are using system resources in the interim.

The second method is to use NOBUF file access whenever possible. When accessing a file in NOBUF mode, the file system does not allocate any file buffers, rather the data is transferred directly from the process's stack to the file. A side effect of NOBUF files is decreased file system overhead. For files with a blocking factor of 1, there is absolutely no difference in program coding to access the files by blocks rather than by records.

System Performance

The HP3000 computer system, and the MPE operating system in particular, were designed for general purpose business data processing. The IMAGE/3000 data base management system, DEL/3000 and V/3000 were all developed to support business transactions. Data entry, online inquiries, and batch processing are not considered to be "real time" operations. And MPE is not considered to be a "real time" operating system. There is no "real" requirement to have a transaction completed in a given time, other than the inherent efficiency of being able to do more work in less time.

In the HP3000 world, system performance is often measured in response time, the time the system takes to process an average user request. There are various ways estimating how long a transaction will take, but in a multi-tasking environment such estimates are often less than accurate. Instead a stopwatch is used, or better yet, the system itself can measure the program's execution time. These measurements take two forms, actual time (wall clock time), and processor time (CPU time). The bottom line is wall clock time, but the two are closely related.

Over the years, the HP3000 computer system has grown from a few users with very small programs, to hundreds of users with extremely large applications. Often, these applications

have out grown the original computer systems purchased to support them in one of two ways. First, as more and more terminals are added to the system, the average response time drops accordingly. Eventually the response time drops below some arbitrary intolerable point, and the system has become too slow. Second, rather than adding more terminals to the system, more functionality is added to the application, causing it to take longer to execute. The result is the same. Eventually the system becomes too slow.

There are three solutions. First, do not add any users or any new features to the system. This is usually out of the question. The second, and perhaps too frequent solution is to purchase a bigger, faster HP3000 from Hewlett-Packard. This could be considered the default solution because it will always work (that is if HP manufactures a bigger, faster HP3000). The third solution is to improve the performance of the existing system and application programs.

In all fairness to the second solution mentioned above, it is very possible that simply buying a faster machine is more cost effective. Programmer's salaries, and program maintenance costs in general are the single biggest cost items in some DP shop's budgets. For a large, stable system, introducing changes can cause more harm than good.

But there is always the next programming project and the next system. The remainder of this article deals with improving system performance by improving application program performance.

Modern day digital computers, including the HP3000, are very fast. They can execute hundreds of thousands, or even millions of operations each second. To the unenlightened, the delay of tens of seconds seems outrageous. What could the computer be doing with all of those millions and millions of instructions? The answer, of course, is "overhead". Time sharing, system intrinsics, COBOL, error checking, formatting, disk accesses, etc... Most of these are out of the control of the applications program, but some of them can be considered in the design of new programs.

The single most obvious contributor to a program's performance (good or bad) is its algorithm. Given any operation to be done, there will always be more than one way in which to to it. Considering the alternative methods, rather than just choosing the first method which comes to mind, can make a major difference.

Consider sorting a set of data. There may be no other specific concept in computer programming which gets more attention in programming texts. Yet all too often, a simple bubble sort is used because other methods are unknown, or are too ambitious for the ordinary staff programmer. The bubble sort could be considered the default sorting technique. There are dozens of different algorithms, from selection sorts to quicksorts. Depending on the application, one of them could be orders of magnitude more efficient than another.

The same can be said for any operation, not just sorting. The solution to performance problems need not be to simply buy a faster processor. An Apple II programmed efficiently can run rings around a Cray-XMP programmed by a neophyte.

* * *

When Hewlett-Packard introduces a new high end HP3000, it usually includes a much faster processor, perhaps twice as fast as the previous high end processor. Yet when an application is moved to the newer and faster processor, CPU time drops, but the wall clock time remains unchanged. The reason is disk overhead. The CPU may twice as fast, but the secondary storage systems are not. In a typical business transaction, many many disk accesses may be required. The processor can handle the data twice as fast, but must wait just as long for disk accesses. The "hurry up and wait" situation is exaggerated even further.

Disk overhead delays are inherent in the mechanical nature of disk drives. The data is stored on rotating magnetic media which passes beneath a set of movable heads. The heads move back and forth across the surface of the disks from the outside edge toward the center of the disks. Each actual head position is known as a track. Modern systems have hundreds of tracks on each disk surface. The disk heads take a finite amount of time to step from track to track. This is known as seek time. A typical seek time for today's disk drives is about five thousandths of a second per track. In addition to the seek time, once the heads are positioned on the desired track, the computer must wait for the requested sector to pass under the heads. This delay is known as latency. Together, these two different delays make up the disk's access time. Once the requested disk sector is passing under the heads, the data transfer takes place in about three ten-thousandths of a second. No matter how fast the computer is, if it's waiting for the disks, then it's waiting for the disks.

With the delivery of MPE-V/P & /E, HP has incorporated system wide global disk caching. Disk caching has been in use for years to increase disk I/O performance. Even the little "E-disks" (E for electronic) used with the HP desktop computers are a form of disk caching. While the MPE implementation does not help when writing to a disk file, it is safe to say that disk data files are read much more often than they are written to.

For applications which use IMAGE/3000, there is little to be said or done about reducing the number disk accesses. Most users could not modify IMAGE/3000 even if they wanted to, and the rest probably would not want to even if they could. But all is not lost. "Ordinary" disk data files, scratch files and tape files are used every day. These files and their access modes are under control of the application program, and their use can be made more efficient.

As mentioned earlier, a disk file's blocking factor can affect performance as well as storage efficiency. MPE always transfers data to and from secondary storage in blocks. If every block contains more records, then fewer block transfers would be required. Remember that the wait for disk time is disk access time, not data transfer time. Transferring a large block takes (effectively) no more time than transferring a smaller block. By doubling the blocking factor, only half as many disk accesses will be required. Likewise by increasing the blocking factor by ten times, only one tenth as many disk accesses will be required.

There are two negative effects of large blocking factors. First, the MPE file system must allocate buffers in main memory for each file opened. If the file blocks are large then the file buffers will be large as well. This could cause memory management problems in extreme

cases. Second, for randomly accessed files, the extra records in the block are not used, and are therefore just wasting main memory.

There are ways to improve performance using random access files as well. As shown above, increasing the blocking factor will not help reduce the number of random accesses to a disk file. However, increasing the number of buffers can. When MPE opens a disk file, it allocates main memory buffers to temporarily store the data between the disk and the user program. Each buffer is large enough to store one block from the disk file. The number of buffers allocated by MPE can be set by specified by using the number of buffers parameter in the FOPEN intrinsic call (and also by specifying the ;BUF= parameter in the :FILE system command). The default number of buffers allocated by MPE is two, which allows MPE to "double buffer" sequential disk accesses. Allocating more buffers for random access files increases the possibility that the requested block will already be in main memory. This is especially true if the same records in the file are being accessed over and over again. Depending on the record size and the blocking factor of the file, up to sixteen buffers can be allocated.

Increasing the number of buffers for a file is still effective under the new MPE-V operating systems with disk caching. Disk caching is precisely the same concept as multiple memory buffers, but on a system wide basis. By locally maintaining file buffers, the system may not have to go looking through the thousands of system buffers.

Another method of increasing the performance of programs which randomly access disk files, is to open the same file more than once. This is particularly useful when part of the file is a header, or directory to the remaining parts of the file. Many of the files used by MPE are of this nature. Program files, SL, RL and USL files, DSG graph and figure files to name a few, all contain directories to the data stored in the file.

* * *

In nearly every discussion of program performance, program segmentation is mentioned. The HP3000 hardware addressing limitation forces programmers to divide large programs into enough pieces such that the individual pieces can fit into memory. These pieces are called segments.

As a program branches from segment to segment, additional system overhead is incurred, especially if the new segment is not in main memory. In this case, MPE Memory Manager activity is required, including a disk access. Memory manager disk accesses are unlike ordinary disk accesses in that they fetch entire program segments, up to 32 Kbytes at a time.

If the new segment is already in memory, then memory manager action is not required, but there is still additional overhead in inter-segment transfers. Eliminating inter-segment transfers requires placing all program modules in the same segment. This is impossible for large programs.

There are several segmentation strategies advocated, and most, or all of them are valid for various conditions. One of the more amusing approaches to segmentation is usually taken by programmers unaware of what segmentation is. It usually involves placing the first N modules into segment 1, the next N modules into segment 2, etc... As programs get larger,

and the programmers become more experienced with MPE, this method usually disappears rapidly.

There is one element of program segmentation overlooked by most discussions of segmentation strategy. Program externals (intrinsics, image, I/O libraries etc.) necessarily reside in external segments. Every time one of these is called, an inter-segment transfer is required.

Some programmers go through great pains to reduce the number of inter-segment transfers, either by re-segmenting their programs, or by modifying the program's algorithm. But only transfers between segments of the program. Transfers to system segments doesn't seem to count. Granted, calls to intrinsics and image libraries usually can't be avoided. But a significant number of calls to system formatting routines could be avoided if the equivalent routines were coded directly into the program. This again raises the question of which is more important: Development time or execution efficiency?

* * *

One of the great mystiques surrounding the HP3000 is the use of privileged mode programming (PM). Referring to the MPE manuals, warnings against the use of PM appear almost everywhere. HP warns that the user's program, the operating system, even the boot copy of MPE on the system disk could be destroyed. True enough, but it would take a conscious effort to destroy anything other than the program's own data, even with PM.

Once the HP inspired fear diminishes, it becomes clear that there are numerous advantages to PM programs. PM allows a program to create and access privileged files, including IMAGE/3000 files. Nearly all of the IMAGE database tools (SUPERTOOL, ADAGER etc.) are privileged programs. By bypassing the IMAGE overhead these programs are able to function far faster than an equivalent program forced to use the IMAGE library.

IMAGE data base files are privileged because of the intricate inter-relationships of the various files and pointers in the files. If they were not stored as privileged files, there would be a tendency for users to access them without the IMAGE library. They could be copied, stored and restored individually, rather than as a set. As privileged files, there is no danger of any of these occurring. Privileged files may be created by anyone who would restrict access to a file (or set of files) to use of specific utilities.

* * *

Another privileged function, which will almost surely not bring the world to an end, is NO-WAIT I/O. In many applications, once some data has been written to an output file, it would be more efficient for the program to go on to the next operation, rather than wait for an I/O operation. A PM program can do just that. For serial types of applications, overlaid I/O and CPU operations can significantly reduce the wall clock time taken by a program.

Another use of NO-WAIT I/O is to control multiple terminals concurrently. Ordinarily, when a read is executed to a terminal, the process waits until the read completes. Using NO-WAIT, reads may be requested to many terminals at once. When data is received from one of them, the process executes the required function, displays the results and executes another NO-WAIT

read. If another read from another terminal completes, the process performs another function.

Using this method, only one process is executing, rather than many identical processes. When accessing IMAGE data bases, concurrent access is not required. Sophisticated locking methods (and their associated overhead) are not required.

* * *

One final advantage of PM code. When using extra data segments for temporary storage, using PM code to move the data back and forth is much faster than using the DMOVIN and DMOVOUT intrinsics. The purpose of using an extra data segment in the first place is for faster access than an external data file. Rather than calling an intrinsic (which checks to see if you have DS capability every time, checks for bounds violations, and then transfers the data), a simple EXCHANGEDB and move instruction are all that is necessary.

Most system managers, for many diverse reasons, avoid PM like the plague. To be sure, there are some malicious people who would forsake all else for the ability to go wandering through the system, doing as they please. But there aren't many of them. Most programmers are relatively professional about their work, and see MPE's privileged mode as a way to write more efficient and/or effective applications.

* * *

There is more than one way to skin a cat. Similarly, there is more than one way to implement a program on the HP3000 computer system. For many "ordinary" or "one-shot" programs, taking the shortcuts provided by MPE defaults is justifiable.

For more sophisticated, production programs, the easy way out may not be prudent in the long run. The prevailing software design methods suggest a very large percentage of development time be spent on the design of the software. This design should not necessarily be constrained to the MPE default way of doing things, merely because it is the default.

Whether it's allocating a scratch file, setting up an IMAGE database, or designing an entire system; knowing the various options involved at each step and choosing the best one, rather than simply using the default, can make the difference between an ordinary product and a superior product.

* * *

[i] It has been pointed out that it may be impossible to purge the user "MANAGER.SYS" since MPE may trap that particular name. If true on a specific release/version/patch, using DEBUG, the actual directory entries may be patched. Consult your HP SE for more information.

[ii] The dichotomy between system security and user friendliness was well illustrated by Steve Johnson in Do You Want to Play a Game?, an article originally published in the Stack (the Baltimore / Washington HP Users' Group Newsletter) and later reprinted in

the SuperGroup Magazine. Steve also described several methods for system managers to reduce the risk of break-ins, specifically referring to modifications to the system catalog file CATALOG.PUB.SYS.

Never Take the Default

#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8638; #O12715 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:19 PM

Programming the New Generation of Hewlett-Packard Digital Computers...

A R.I.S.C. Tutorial

Second Edition

by Jeff Hecker

Apogee
4632 W. Frankfort Drive
Rockville, MD 20853

Preface to the Second Edition

This paper began in 1985, when non-ignorable rumors concerning Hewlett-Packard's new "Spectrum" series of computers began in earnest. At the time, being from a hardware background, I was asked by some, what all of the fuss was about In February 1986, after HP's formal announcement of the HP3000 series 900 computers, the first edition of this paper was written and submitted to the 1986 INTEREX conference. During the intervening time, much has been learned about the new systems. Between the time this paper is written and the time it is read, it is likely that HP will have shipped series 900 HP3000s. With this in mind, much of the speculation of the first edition has been deleted, since it won't be speculation at the conference.

Programming the new Generation of Hewlett-Packard Digital Computers... A R.I.S.C. Tutorial

by Jeff Hecker
Apogee

After all of the gossip, rumors, guesses and second guesses, Hewlett-Packard has announced the first products to come from the Spectrum project. Despite all of the claims of compatibility, many people remain unconvinced. How can a completely new machine really be compatible? How can a completely new machine with a completely new instruction set be able to execute old programs? How can programmers take advantage of the new instruction set for highest possible performance?

The new HP computers follow a recent trend in the computer industry towards what are known as Reduced Instruction Set Computers (RISC). RISC computers in one sense, might be considered to be a throwback to the very first days of programmable computers. Then, there were only a very few instructions such as load, store, add and compliment. Not even a subtract instruction! To subtract two numbers, the programmer would fetch the first operand; then fetch the second operand; compliment the second operand; add it to the first operand; and finally store the result back into memory.

In this case, a simple operation took twice as many instructions as it would first seem to require. Also these were the days of machine level and later, symbolic assembler programming. There was no such thing as a FORTRAN or COBOL compiler. It became clear to computer designers that it would be much more efficient if the computer could perform more complex operations. This way, the number of instructions coded by the programmer could be reduced. Fewer instructions to be coded translates directly to lower costs and to higher software reliability. (Sound familiar?) The current generation of Complex Instruction Set Computers (CISC) was born.

These were the days of slow, expensive memory. Anything that could reduce the size and/or the speed requirements of the memory system was implemented. CPUs were expensive, but there was only one CPU. There were thousands, or tens of thousands of memory circuits. As instructions became more sophisticated, fewer and fewer were required, reducing memory costs dramatically.

Over the years, instruction sets continued to expand. Not only could computers subtract, but they could multiply and divide too. Sophisticated operations such as memory to memory copy, floating point math, iterative loop instructions, and even memory block checksum calculations were incorporated into the computer's instruction set. When microprocessors were introduced in the early 1970's, one measure of sophistication was the number of different instructions they could execute.

Over those same years though, software technology was improving as well as hardware technology. FORTRAN followed by COBOL, PL/I, Pascal, C, Ada and others have been

developed to further improve the efficiency of the programming task. Programmers no longer had to worry about which particular machine instruction was being used. In fact, today's programmers are so thoroughly insulated from the actual computer which executes their programs, that one actually thought that a FORTRAN formatted WRITE statement was carried out by hardware.

High level languages have caused two sweeping changes in the computer industry. First is program portability. Since a programmer (and therefore his program) are no longer concerned with the actual computer instructions, he can take his program to a completely different computer with (in theory) no changes.

The second change was much more subtle. With the use of high level language compilers, a very few people decide which of the computer's instructions actually are ever used. When a compiler is written for a particular computer's instruction set, it must translate the language source statements into the appropriate sequence of machine instructions which will accomplish the needed function. Once the compiler is written, only those instruction sequences generated by the compiler will ever be executed.

Over the years, compiler writing has advanced as well. Compiler writers no longer just arbitrarily chose instructions from the CPU's instruction set. Execution speed is also being considered. The classic example is the case of the VAX loop instructions. They are expansions on the standard "Decrement and branch on not zero" instructions found in many CPUs. But they executed so slowly, that compiler writers issued more traditional sequences of simple instructions which executed faster. This instruction (and others like it on many different computers) are the excess baggage that CISC computers must carry around with them when a new processor is designed. Many of a CPU's instructions may be rarely, if ever, executed. This is particularly true when one group designs the hardware, and an independent group develops the software and compilers.

This phenomenon has not gone unnoticed by the computer designers. Beginning in the late 1970s, several researchers began to independently measure which instructions were actually being executed by CISC machines. Their results were another variation of the 80-20 rule. 80 percent of the time, computers were executing the same 20 percent of their instructions.

At the same time, another phenomenon was beginning in the computer industry: the UNIX operating system. UNIX and its native programming language "C", were appearing on more and more computer systems every day. Since all of these UNIX systems came from one source, they all used precisely the same C compilers. Suddenly there were hundreds of machines executing the same 20% of their instructions 80% of the time.

These developments triggered two reactions from computer designers. One was the C-machine. The C-machine was designed specifically to execute C programs. Its instruction set was taken from those actual instructions generated by the UNIX C compiler.

Secondly, academic researchers considered the idea of eliminating the rarely used instructions from the computer. Taking the results from their measurements, they determined which instructions were actually useful, and which were not. Many of the instructions which were

relatively unused are also the same complex instructions which have been added over the years. RISC machines were born.

RISC machines offered great potential benefits to computer circuit designers. Eliminating some (or all) of the more complex instructions allowed circuit designers to build a faster system for an equivalent cost. Fewer different instructions also require fewer different CPU components. Fewer CPU components require fewer transistors to implement them. Fewer transistors require less silicon per chip produced. Less silicon per chip results in a higher yield. Fewer transistors also require less power. Lower power increases reliability. There is almost no fundamental reason not to implement a RISC CPU.

Recently, several computer manufacturers have introduced RISC CPU based systems to the market. The Bolt, Berenek & Newman (BBN) C-70 is a C-machine, designed specifically to run the UNIX operating system. Pyramid's 9X systems are VAX-class minis designed for the general market. Several other RISC machines have also been announced and delivered.

In Hewlett-Packard's case, simply implementing a new RISC based computer system would have alienated a large potential customer base. Not only must the new generation of HP RISC based systems have more power than the current HP 1000/3000/9000 systems, but they must be compatible with these systems as well.

There are several different approaches to compatibility. The simplest is to require each application program to be re-compiled on the new system. Another method is to actually include parts of the old machine in the new machine. The third approach is to emulate the old machine with the new machine.

The first method is in common practice in the UNIX community. Compatibility is defined at the source code level. It is not unusual to require the complete re-compilation of all programs even when an O.S. update is installed. This is particularly likely on some UNIX look-alike systems such as Microsoft XENIX. Fortunately, UNIX provides many more flexible and automatic program generation tools than any other operating system, making complete re-compilation of any program very simple.

The second approach was used by Digital Equipment Corp. when their new 32-bit computer system, the "VAX" was first developed in 1978. At the time, the PDP-11 was widely installed, and DEC had the same problem which HP has had to face. DEC's solution was to include a PDP-11 processor as part of the VAX processor. DEC customers could run their existing PDP-11 programs on the VAX unchanged, until the programs were converted.

The approach chosen by HP is to emulate the HP3000 instructions in software. A program will examine each HP3000 instruction, and branch to the appropriate subroutine for execution.

Of the three possibilities, the first seems least expensive, fastest to implement, achieves the highest performance, but requires the most effort by customers making a conversion. Also, each customer must have the source code for each of their programs. This is not possible when customers buy their software from companies such as HP. The software developer is,

therefore, required to convert all of the software before the system can be delivered, yielding a false economy.

Including an HP3000 processor in the new machines would be contrary to the entire RISC philosophy. The cost of the additional circuitry, and additional power consumption would be passed along in each unit, even after conversion to the new system. However, this method is most reliable in the sense that there are no new compilers or interpreters which might not function properly in every instance. This is why DEC chose this path.

An interpreter is HP's compromise between the hardware integrity and having to convert the entire HP applications software catalog. An interpreter can also execute a program for which the source code has been lost, or a program which has been patched for a particular configuration. The other side of the coin is the severe performance penalty of interpreted operation.

- - - - - - - - -

How do instruction set emulators work? How can one machine execute another machine's instructions? Why is there a performance penalty, and how severe is it?

The new HP3000 models from the Spectrum project will be able to execute existing HP3000 programs in "compatibility mode". Since the new computer system uses a new instruction set, the CPU itself will not be able to execute the instructions. A system utility will be invoked transparently to the user in order to execute the program in HP3000 compatibility mode. This utility is the emulator or interpreter.

The interpreter is a Spectrum native mode program which uses the HP3000 program as input data. Consider the BASIC/3000 interpreter currently available. BASIC is an HP3000 "native mode" program which uses a BASIC program as input data for execution. The principle is precisely the same. The HP3000 interpreter is a Spectrum native mode program which will examine each HP3000 instruction to be executed, and then execute it.

Interpreters, by their very nature, incur a tremendous amount of overhead. But how much? Consider the steps which must be carried out by any interpreter for each emulated HP3000 instruction:

1) Check that the instruction pointer does not point beyond the current code segment limits. (In practice, by properly aligning and filling HP3000 segments into HPPA pages, this step might be performed by hardware during the following step.)

2) Fetch the instruction to be emulated from memory.

3) Increment the program instruction pointer so that the next instruction can be fetched from the correct memory location.

4) Determine which instruction this one is, either by look-up table (a 65000 item table); or by examining the individual bit fields within the instruction word, which might take several steps.

5) Branch to the appropriate emulation subroutine to execute this instruction.

6) Execute whatever sequence of native instructions is required to achieve the same result as the emulated instruction.

7) Branch to (1) to fetch the next instruction.

These are the very same operations which every HP3000 computer must use to execute a program. When implemented in hardware (such as an HP3000 CPU or a Spectrum CPU) several of these steps can occur simultaneously in different parts of the CPU. When emulated in software by a single processor CPU, each of these operations must occur sequentially one after another. A rule of thumb is that the interpreter overhead will cause an order of magnitude performance difference.

Consider the HP3000 NOP (No OPeration) instruction. Typically, a computer can execute a NOP as fast or faster than any other instruction. In the emulator example above, the emulator will have to execute 6 native mode instructions to emulate a NOP. We'll assume that the emulated NOP (step 6) is skipped. This yields a best case expected performance of about 1/6th of an equivalent native mode program.

Of more concern is the expected performance of real instructions, not NOPs. Most of the instructions in an HP3000 program are LOAD and STOR. These instructions typically reference either the DB or Q registers. To emulate a rather complex LOAD Q+5,I,X instruction, some relatively sophisticated code is required. Remember that the +5 is embedded in the instruction word, not in a separate variable.

5.1) Extract the offset (+5) from the instruction into a scratch register.

5.2) Add the value of the Q register pointer to the offset in the scratch register.

5.3) Is this an indirect access (,I)? If not then branch to (5.6).

5.4) Check for a bounds violation. This actually involves two steps. Is the referenced location greater than or equal to the TOS pointer register? Is the referenced location less than or equal to the DL pointer register? If so, then trap to a bounds violation routine.

5.5) Fetch the data from the referenced memory location into the scratch register.

5.6) Is this an indexed access (,X)? If not, then branch to (5.8).

5.7) Add the value of the X register to the scratch register.

5.8) Check for a bounds violation.

5.9) Double the value of the scratch register because this is a word fetch, not a byte fetch.

5.10) Fetch the data pointed to by the scratch register into the scratch register.

5.11) Check that adding 2 to the top-of-stack (TOS) pointer will not cause an HP3000 stack overflow.

5.12) Add 2 to the TOS pointer register to make space for the LOADed data.

5.13) Store the data in the scratch register to the location pointed to by the TOS register.

5.14) Return to the main emulator loop.

This code sequence could be typical for emulating an HP3000 instruction. There are less sophisticated instructions. For example the stack operations such as ADD, SUB, DUP and DEL are not nearly as difficult to emulate as LOAD and STOR.

There are also more complex instructions which are executed frequently such as conditional branches, MOVE, SCAN and PCAL. PCAL (Procedure CALL) is particularly difficult. It has been stated several times that a program will be able to execute in compatibility mode and native mode and be able to switch back and forth. It has also been stated that several MPE intrinsics will not be rewritten in native mode until future releases of MPE-XL. When the HP3000 emulator encounters a PCAL instruction, not only will it have to route through the internal and external STTs, it must also determine if the called routine is in native mode or not. If it is, it must actually call it, otherwise it must continue emulating the HP3000 instructions.

MOVE and SCAN must actually execute as a loop in the emulator. These instructions will incur the usual loop overhead plus bounds violation checking during each loop iteration. In some cases, an optimized emulation can test for bounds violations for the first and last locations accessed, rather than once per memory access. The exception is the HP3000 SCAN WHILE and MOVE WHILE instructions. The emulator has no way to determine how many locations will be accessed, and has no alternative to checking for bounds violations on each access.

Overall, using a software emulation approach to compatibility provides the capability to execute any HP3000 program on the Spectrum class machines. No re-compilation is necessary. The interpretation penalty is partially compensated for by the increased processor speed of the Spectrum machine.

The software emulation approach offers many other potential capabilities as well. The HP3000 instruction set contains no particular magic which makes it simple or efficient to emulate. Virtually any instruction set may be emulated in much the same fashion. The possibilities are numerous; An Intel 8086 emulator for MS-DOS programs; An IBM System/370 and/or XA emulator; or a DEC VAX emulator able to run the VMS operating system. All of these have been alluded to by various people associated with HP's Spectrum project.

The actual expected performance will depend on the ratio of compatibility mode code vs. native mode code. A typical HP3000 application is based on IMAGE data bases. Most of the time spent in such a program is spent executing IMAGE system intrinsics. There are not many instructions between calls to DBFIND, DBGET, and DBPUT. If the IMAGE intrinsics have been rewritten in native mode, then a program which spends most of it's time in IMAGE code will execute nearly entirely in native mode. A good analogy would be a

BASIC/3000 program which accesses an IMAGE data base. Using the BASIC compiler does not affect execution time very much because most of the time is spent in the IMAGE intrinsics.

Emulation or compatibility mode is only used until programs are converted into native mode. This is the case for MPE-XL code, intrinsics, HP system and applications programs, and for users' programs. Once everything is converted into native mode, compatibility mode will be obsolete.

The main argument against RISC computers has always been that they require many more instructions to accomplish a given task. Without improved compiler technology, this is true. Since each instruction gets less work done, more instructions are needed. Even though each instruction executes faster, there could be many more instructions to execute. More instructions also occupy more main memory. The success of a RISC based CPU depends on it's compilers. Incompetent compilers will doom any RISC based system to failure.

The compiler should be able to generate an efficient sequence of instructions, rather than just any old sequence. Often the compiler will make another pass (or passes) over the code in order to eliminate redundant instructions and other inefficiencies. This process is known as optimization. The current HP3000 compilers have no optimization pass, and in fact generate pretty sloppy code. The goal of an optimizing compiler is to generate efficient code. For now, "efficient" can be defined as compact, fast, and of course correct.

No one has seen HP's optimizing compilers yet, so there is no way to estimate how well they do the job. Most compiler optimizations have been well known and understood for quite a few years now. There's no reason to believe that the HP compilers will not do a good job optimizing program code.

But how can a programmer get the maximum possible performance from HP's new Spectrum computers? As always, programming in assembly language yields the best possible code, based on size and speed. The penalty for programming in assembly is severe. Programmer productivity is very low, and the chance of errors going un-detected is much higher than with a modern language such as PASCAL. Still, there are a few things to consider when programming in a high level language.

The HP3000 was based on a stack architecture with very few usable registers. HP's Spectrum computer systems are register based. Every time a variable is referenced, it must be brought into a register before it can be operated upon. If a program uses a large number of variables, they must be constantly moved back and forth from memory into CPU registers. Re-using variables for more than one section of code will allow the compiler optimizer to keep those variables in registers longer. Consider the following PASCAL examples:

```
{ swap some arrays }
for i := LOW to HIGH begin
    temp1 := a[i];
    a[i] := b[i];
    b[i] := temp1;
    temp2 := x[i];
    x[i] := y[i];
    y[i] := temp2;
    end;
```

In this example, the compiler needs the variables I, LOW, HIGH, TEMP1, TEMP2, and pointers to the arrays A, B, X, and Y. These will require 9 registers. This is a trivial example, and in fact all of these variables could be loaded into the registers of the Spectrum machines. But it shouldn't be too hard to imagine a more realistic case where the compiler is forced to move variables in and out of registers too often. An optimizer can't easily tell which variables are important and which are scratchable, so it must save them all. (In fact, depending on the effort expended by HP, it is possible for the compilers to look far enough ahead in a program to determine if a variable can be scratched, but it is a non-trivial procedure.)

Rewriting the same loop to use fewer different variables can reduce the register requirements of a program segment. Consider this rewrite of the previous loop:

```
{ swap some arrays }
for i := LOW to HIGH begin
    temp1 := a[i];
    a[i] := b[i];
    b[i] := temp1;
    end;
for i := LOW to HIGH begin
    temp1 := x[i];
    x[i] := y[i];
    y[i] := temp1;
    end;
```

Now only I, LOW, HIGH, TEMP1, pointers for A and B are needed at one time. This is a reduction from 9 registers to only 6. It is possible that in some situations, the duplicated loop overhead is less than the register load/unload overhead. If a block of code requires more registers than are available in the machine, then the compiler must swap the variables out to memory when not in use. Although the HP Spectrum machines have thirty-two registers, they are not all available for use within a program. Some are reserved for milli-code operations, some are reserved for the HP3000 emulator, some are reserved for procedure parameter passing, and some are reserved for CPU overhead such as stack pointers and program pointers.

Register allocation has never been a concern on HP3000 computers before because there were no registers to allocate. Every local variable is just as accessible as every other variable. In HP3000 promotional literature, this was touted as an advantage of the stack architecture.

Each local variable could be accessed directly. But each variable access requires an indexed memory operation (ie Q+5). Register to register access is much faster.

- - - - - - - - -

Another feature incorporated into the Spectrum architecture is pipelined instruction execution. It is possible to begin an instruction before the previous instruction has finished executing. This may not seem to make any difference (and it doesn't when programming in a high level language), but it must be considered when programming in assembly language, or when writing language compilers.

Consider the actual operations of fetching an instruction from memory, decoding it, and executing it. These three steps require three sets of hardware.

T+0) Generate the memory address of the next instruction, and read the contents of memory into a CPU register.

T+1) Decode the instruction by routing the appropriate bits into gating and latching circuits.

T+2) Those circuits which were activated now actually execute the instruction operation, perhaps requiring access to main memory.

T+3) Main memory access for those instructions which require it. Otherwise begin a new instruction.

In non-pipelined machines such as the HP3000, each step would execute sequentially to process an instruction, and then repeat. In more recent CPU designs, including the HP Spectrum computers, pipelining attempts to reduce or eliminates each component's idle time. In this example, each CPU stage is idle for 2 out of 3 time cycles. By beginning the next instruction before the current instruction has completed, this idle time is eliminated, and throughput is increased. Consider the pipelined system in the figure below. The boxes represent the work being done by each processor stage during a given time period. Time is represented by T+n where n increases. The instructions are indicated by the memory location they are fetched from; M+n where n increases.

	Non-Pipelined CPU			Pipelined CPU		
Time	Fetch	Decode	Execute	Fetch	Decode	Execute
T+0	M+0			M+0		
T+1		M+0		M+1	M+0	
T+2			M+0	M+2	M+1	M+0
T+3	M+1			M+3	M+2	M+1
T+4		M+1		M+4	M+3	M+2
T+5			M+1	M+5	M+4	M+3
T+6	M+2			M+6	M+5	M+4
T+7		M+2		M+7	M+6	M+5
T+8			M+2	M+8	M+7	M+6
T+9	M+3			M+9	M+8	M+7

Of course pipelining is not quite as simple as this graph would imply, but it does give the right idea. In the non-pipelined system, the entire CPU must wait for the completion of each instruction. In the pipelined system, more of the CPU is kept busy more of the time. The pipelined processor in this graph is not faster than the non-pipelined processor, but at the end of a given period of time, it has executed more instructions. If each time period in the graph were one microsecond, how would each processor be rated for speed? The non-pipelined processor executes 1/3 MIPS (Million Instructions Per Second). Rating the pipelined processor is much more difficult. There is no correct rating. Any of 1/3 MIPS, 1.0 MIPS, or "up to" 1.0 MIPS might be used. It will depend on the method chosen by the manufacturer.

The real world is also a bit more complicated than pictured here. Also, this example of pipelining shows no provision for interrupts. Where does the processor restart after the interrupt? After the instruction which has most recently been fetched? Or after the instruction which has finished execution? What if some stages perform partial instruction execution? Some pipeline implementations have 5 or 6 stages, not just the three shown here. The additional stages are used for indexed memory accesses and for faster memory management. In this example there is no mechanism for accessing main memory during the instruction execution phase. If the execution stage requires access to main memory, then the instruction fetch will be forced to wait.

This is another advantage of the register based CPU. As the ratio of register to register instructions increases, so does the pipeline efficiency. The more often the processor must access memory, the more often the pipeline will be kept waiting.

- - - - - - - - -

All in all, time will tell how successful the new HP3000 high end computers will be. Many of the new architectural features of the Spectrum processors are clearly included so that HP need never again say it's sorry (particularly the 32 bit plus 32 bit address capability). The HP3000 64 Kbyte data address limit has not been winning many latter-day followers in the light of competitive minicomputers.

Hewlett-Packard has not leap-frogged anybody with their new computer systems. But with new high performance, pipelined processors, large memory capacity, optimizing compilers, relational databases, true virtual memory (VM), symbolic debuggers, and HP3000 family compatibility, HP is finally keeping up with the Jones's.

- - - - - - - - -

There are two very good books available which describe the trials and tribulations of designing a new computer system. They can provide a view of the world from the other side of the fence. New computer systems don't grow on trees. Designing, building and programming them is a monumental task. One of the books is The Mythical Man Month (Fred Brooks). Fred Brooks was the project manager of the IBM OS/360 project. The book describes some of the things which happened during that project, and the lessons learned. The Soul of a new Machine (Tracey Kidder) chronicles the group of engineers who designed the Data General Eclipse computer systems. Both books are very good reading for a project manager, or for someone who has had to put up with project managers. Both are commonly available at public libraries and bookstores.

Jeff Hecker has been involved with the HP3000 and the MPE operating system for the past eight years. He has been responsible for moving, debugging and optimizing several applications from the HP3000 to other computers and operating systems, and back again. Prior to working with the HP3000, Jeff was a digital hardware design engineer for a Maryland based telecommunications company.

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM

#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM
#S'8639; #O12716 * MS4337, MANAGER.SYS; MARK * TUE, MAY 12, 1987, 9:20 PM

ABSTRACT

When the Systems Tables Manual is not Enough: Bit Picking in Application Data Files

Jeff Hecker

For many years Hewlett-Packard provided only enough HP 3000 software to allow users to write and compile their own applications. This was (and is) the Fundamental Operating Software (FOS). Along with FOS, customers could obtain the System Tables manual which descibed the memory and disk file formats used by FOS.

In recent years, HP has ventured into the end user application software market, with products such as manufacturing, accounting, communications, office automation, and graphics. But there is no equivalent to the tables manual for these products. There are times when, for whatever reason, the information in these files is needed in a way not provided by the HP product.

This article describes the general methods used to decode various files, and the results obtained in one case study of HP figure Files. The material contained is on a relatively technical level, but may be useful to those with only a casual interest in knowing how to look into these files.

A Mechanic's View of Turbo

Dennis Heidner
Boeing Aerospace Company

Abstract

HP released TurboIMAGE in 1986, and the general public has since had the opportunity to read the product specifications and marketing literature. Finally we have another view, this time not from the vehicle's designers but from the mechanic next door. This paper reviews some of the practical guidelines that have been developed for IMAGE/3000 applications and how they have changed with TurboIMAGE. I present techniques that were used to dismantle the database engine, see what makes it run and discover what clogs it up.

Introduction

My co-workers (and family) described me as "a little boy waiting to open up his Christmas presents". I felt a more appropriate description was a Car and Driver test driver waiting for the new model year's cars to arrive. In any case, it had been several years since the TurboIMAGE product had been "announced", and now I had my chance to test drive it!

Visual Inspection

My visual inspection of the new engine revealed that, for all practical purposes, the appearance had not changed. That was pretty important since I was concerned about how it would work with the rest of my vehicle (application software). The owner's manual was rewritten to reflect the new powerplant. However, a number of small errors crept into the manual. (Some of those errors later would prove to be fatal to some!) A new accessory called DBCONV was provided. This accessory made the installation of the engine

quite easy. The controls for the engine were the same, with a few new additions. For example, DBUTIL now included the following new commands: [1]

Command	Function
Enable x for AUTODEFER	Automatically use DBCONTROL mode 1 AND allow multiuser access!
Enable x for ROLLBACK	Turn on greatly-enhanced transaction recovery
MOVE	Move a specific dataset (use filename) to a specific disc drive
SHOW x DEVICE	Show on which devices the datasets are

The TurboIMAGE program DBRECOV had also been modified to support the new recovery rollback options and support MPE user logging enhancements.

A Closer Look

I had a pretty good idea what to expect even before the TurboIMAGE specifications "leaked out". You see, HP had previously introduced a version of IMAGE for the scientific HP9000 series 500 computer. While HP was telling the world about their great new scientific computer, they were also selling their concept of a factory network. It was reasonable to assume that the same capabilities of the 9000 would make it into the "top-of-the-line" HP business computer!

I am sure that some of my local HP SE's thought that I had some internal spy network in order to get the changes before they were announced to the general public. Well, my secret is out. I read my SSB's, Communicators and HP product announcements for similar product lines.

In case you have missed them, the specs are: [2] [3]

	IMAGE/3000	TurboIMAGE
Data items per database	255	1023
Data items per data entry	127	255
Data sets per database	99	199
Detail sets per master	16	16
Master sets per detail	16	16
Maximum entry size	4094	4094
Entries per data set	8,388,607	2,147,483,647
Entries per chain	65,535	2,147,483,647

(The 2,147,483,647 entry limit may never be reached on the current hardware and version of MPE because of hard-coded file system limitations).

Out on the Test Track

Since I want to uphold my Car and Driver image, let me describe the test track that I had set up and the performance I saw with the new engine. Then I will cover the engine tear-down, what I found, my likes and dislikes.

The equivalent of test tracks for computers are benchmarks. There is a small problem though: benchmarks should be considered a four letter word. No two benchmarks are alike, and seldom do they match reality. Trying to use the results of a benchmark to predict performance of a computer is similar to using the EPA estimate for gas mileage for a new car: lots of luck! After considering my options, I decided to stay with a benchmark that I had designed seven years earlier to test transaction logging. This battery of tests is not the greatest, but it is the only set of tests that I knew of which had been used on every version of IMAGE since the early release of IMAGE-B.

The test consisted of a stream job and several programs which would make DBPUTS, DBDELETES, DBFINDS, DBGETS and DBBEGIN/DBEND transactions on a small database. The stream job was written so the same transactions were repeated with all TurboIMAGE database recovery options and AUTODEFER. The same stream job (minus the test for AUTODEFER and Rollback) was used on IMAGE-B.

The results were apparent in a matter of hours. You see, the test would take about one hour when run with IMAGE-B, but when we ran the same test on a TurboIMAGE installation, the test never completed. Not too bad for a faster engine! It turned out that the problem was not a slower engine or some obscure bug but that Turbo was working just the way it should. My benchmark, which worked fine for IMAGE-B, was dependent on a slow, serially-threaded IMAGE,

but with those restrictions removed, the portion of the benchmark which said it was time to stop never reached its limit condition!

Okay, I know that's a bit confusing, so let me explain further. The benchmark had a master process which would make the timing measurements, while several son processes were trying to make hundreds of DBUPDATEs against the database. The idea was to simulate a transaction-intensive environment. Logfile analysis showed the problem was that IMAGE-B did not honor MPE priority scheduling. Instead, the processes ended up in a round-robin-type of priority, regardless of the CPU time each used. This was contrary to the MPE scheduling philosophy, which is to penalize heavy resource users in favour of short easy-to-handle transactions. With TurboIMAGE, the cheating stopped. The result, on Turbo, was that my main process, which happened to consume a lot of CPU time and other critical resources, was dropped in priority, while the "loading processes" were always allowed to run because they required very little CPU time. The main process was coded to determine when the test was complete, but it was never completing because it had a low priority. This may sound like there were no transactions being made on the database, but the opposite was true. With IMAGE-B, several thousand transactions would have been written to the logfile. With TurboIMAGE, after allowing for the longer run time, the number was in the tens of thousands! In the end, I modified the benchmark in order to run the test on TurboIMAGE. The results are shown in A-1, at first glance not very impressive. However the test compare the best case of IMAGE/3000 against the worst case for TurboIMAGE! (Remember IMAGE/3000 was serially threaded, in effect tuned for a single user, while TurboIMAGE flies when there are concurrent users.) Chart A-2 is a comparison between DBPUTs with ILR and logging (The TurboIMAGE logging test are with AUTODEFER).

Tools for Dismantling the Engine

Now that we've taken the change for a test drive, no mechanic could resist the chance to kick a few tires and get out the old toolbox to see how the new thing works.

Memory Dumps

Some of my more exciting nights were spent reading memory dumps. I am sure my wife and Marguerite Russell thought that the strain of it all had gotten to me - there was a grown man laughing and cheering at a bunch of octal numbers. I assure you, I really am quite normal, but I could not restrain the excitement generated when I saw that HP had corrected, enhanced or implemented many of the major complaints or wishes. The memory dumps were useful in

determining how TurboIMAGE was handling its extra data segments (XDS's). For instance, how is the database system control block (DBS) used? It was through the reading of memory dumps that I came to believe the purpose of the DBS had changed from what HP had previously documented. [4] You see, they claimed that all access to the database would be coordinated through the DBS. If that was true, the DBS should always be present because of the frequency at which it is accessed. What I had observed is that the DBS appeared to be always absent. The memory dump was the first place I was able to confirm the use and operation of the GLOBAL AFT. Finally, with a little bit of luck, I was to duplicate some of the system failures that were reportedly caused by TurboIMAGE. Then by matching up the failure with the known problem report in the SSB or patch summary, I obtained a better understanding of what was occuring internally in TurboIMAGE.

I-dump Files

One of the (hopefully) seldom-used features of IMAGE implemented in all versions is a feature called the IMAGE INTERNAL INCONSISTENCY ABORT. Whenever IMAGE or TurboIMAGE detects something that looks strange, (besides my designs), it creates an optional snapshot file and aborts the offending program. Generally, this abort only occurs if a broken chain or damage to the database is encountered. The function of the snapshot (I-file) is to aid the IMAGE lab at HP locate and correct bugs. These I-files are also useful to a mechanic trying to figure out how an engine works.

Unlike the memory dump, the I-file contains the user's stack and all Turbo control blocks that the process needed. These are typically the stack, database global block (DBG), database buffer control block (DBB), database user local control block (DBU) and the intrinsic level recovery control block (ILCB).

SOOT/ADPAN

One of the fancier toolsets in my mechanic's box is a pair of programs called SOOT and ADPAN. SOOT is a program, written by Ben Norton of Boeing Computer Services, which allows a user to capture the program stack for any process. [5] ADPAN allows a user to analyze this "captured stack" (dump file) and look at the stack markers, files which were open, data values and program global data. [6]

Using these tools, several interesting points were immediately visible. The first was that HP has created a number of new internal procedures which implement TurboIMAGE's new buffer management algorithms. (See A-3,4,5 & A-8,9.) The second was that FOPEN's

for individual datasets are no longer recorded in the process stack (this may not be news now, but it was in 1986). (See A-6,7) Finally, the meaning of the BASEID, the first word in the base parameter of a DBOPEN call, had changed. Previously, with IMAGE-B, this word had two meanings. The upper six bits contained an accessor count and the lower ten bits contained the DST number of the DBCB. The increased table sizes allowed by MPE V forced this to change. The problem was that in ten bits the largest number you may have is 1024, but MPE V now supported 4096 XDSs. The HP literature stated that the BASEID would be used as an index into the DBS, which in turn pointed to a new index for the DBCB or DBG. That does not seem like a very efficient process; in real life, HP has done some fine tuning. The BASEID is now the DST number for the DBU. The DBU, not the DBCB, has become the starting point for all Turbo intrinsics!

Debug/Decomp

Debug and Decomp were tools of last resort. They are similar to a mechanic's modifier and eliminator (also known by the generic terms "hammer" and "blow torch"). The use of DEBUG/3000 was risky, since I had to be running in privileged mode. It did allow me to single-step my way through some sections I was perplexed by; in other cases, I was able to single-step my way to unintentional system failures! Decomp allowed me to view the only "source code" available, the run-time library. Several interesting observations were made with the use of Decomp, including new states for the rootfile (these were later explained by Doris Chen's rootfile paper[7]). Decomp and Debug were the only real tools available when trying to figure out how the new buffer management algorithms work.

Trace Files

HP responded to the frequent user requests for a database performance monitoring tool by developing the database PROFILER. This package must be bought (like OPT and APS). In theory, it will allow the Database Administrator to monitor the frequency of accesses to various databases, how the databases are being accessed, how efficently the internal buffers are being used and whether or not the database needs restructuring. When IMAGE-B was rewritten into TurboIMAGE, many of the internal procedures were instrumented. This means that special hooks were added to collect information about how, when, where, and what called the procedure. This information is collected in a XDS or written to a file. The PROFILER "Turbo Data Analysis" routine is later used to replay the information which was collected and generate the profiles for your databases. While tinkering with the system, I discovered that it was possible to turn on the collection mechanism without having the

profiling package installed. The resulting Turbo trace file was a important in understanding how Turbo now works. (See A-8,9.)

What's Under the Hood

Rootfile Changes

Understanding the rootfile turned out to be a tedious task. Even with my previous knowledge of the IMAGE-B rootfile, decompiling and understanding the new regions consumed several weeks. In the end, I was quite proud of myself. I had done it! Then, two weeks later, I received my copy of the Madrid conference proceedings with Doris Chen's article about the rootfile. Her paper included a copy of the rootfile layout as an appendix! [8][9] (Oh well, so much for my summer tan...)

Since I spent so much time figuring out how to decode the rootfile, I might as well write one paragraph on how to do it. Here we go! The first step is to choose a simple database, (I chose the SAMPLE database from The IMAGE/3000 Handbook). Then you must compile it with DBSCHEMA on an IMAGE-B system. You may dump the rootfile in its entirety using DBDUMP. The next step is to create the database using DBUTIL, then (you guessed it) dump the rootfile and associated sets using DBDUMP. After you have a view which shows the locations of items/sets with IMAGE-B, you repeat the process for TurboIMAGE.

Control blocks

The most significant enhancements made were the restructuring of the database control blocks. With IMAGE-B, the two main players were the DBCB and the ULCB. In effect, whenever an IMAGE intrinsic was called, the user's program obtained exclusive control of the DBCB, copied in user-specific information from the user's stack then modified the DBCB (by copying in information from the ULCB). The assembling of buffers and user logging information was performed in a trailer area at the end of the DBCB. The net effect was that only one process could access the database at a time (the access time was typically in thousandths of a second).

TurboIMAGE addresses this problem by splitting the DBCB into two new control blocks, the database buffer (DBB) and the database global (DBG) blocks. Most of the processing performed previously on the DBCB was moved over to the ULCB, which was renamed the database user (DBU) block. With TurboIMAGE, as much of the preprocessing and assembling of information as possible is done in the user-specific control block. This means that the DBG is only

required when the database is opened, a lock is acquired or released (DBLOCK/DBUNLOCK) or global information is requested. The DBB is now only needed when we are trying to move information from the the physical media (disc drive) to our user-specific DBU. Even then, TurboIMAGE has been enhanced so that it tries to eliminate as much contention as possible. This was accomplished by allowing TurboIMAGE to place a "hold" on single buffers, not the whole DBCB.

The function of the RDBCB and ILCB remain basically the same as before. The function of the SDBCB (now the DBS) appears to have changed. From what I was able to glean from the tear-down was that the DBS is now used primarily to coordinate PROFILER operations.

Turbo also introduced several new control blocks. These are the database extension (DBX) block, used to hold the overflow from the DBG; the database trace (DBT) block, which appears to be used to collect and coordinate Profiler activity on a database level; and lastly, I saw references (via DECOMP) to another control block which I will call the database mystery (DBM). As you may have noticed, all the control blocks have three-character acronyms; that's not necessarily HP's convention, but rather an author's privilege I've taken. [10]

The dividing of the functions of the control blocks means that we now have more area for locks, more and larger buffers and increased numbers of items and sets. The flip side of the issue is that we now will have more, larger DSTs. Larger, of course, means we will consume more resources than before.

Locking

Recently, how to lock has been a rather controversial issue. There appear to be as many "right ways" to lock as there are files in PUB.SYS. One thing is sure though, the increased concurrency that is now allowed with TurboIMAGE spells BIG trouble if you have a weak (or nonexistent) locking plan. HP also has recognized this and increased the region available for locks from 6k words to 8k words. [11][12] If you wish to take advantage of the new recovery tools available, or if you wish eventually to migrate to "ALLBASE" then you must consider implementing the HP-suggested method of strong locking. [13] Strong locking is intolerable if you use database level locking (some third party vendors still do!). However, there is really no need to resort to database locking, because HP has provided us with a predicate locking capability. By using predicate locking, you may acquire locks on specific item values, many different datasets (including masters) or the entire database. The use of dummy datasets, extra databases and some of

the other gimmicks will probably cause you more pain than it will actually be worth in the end.

Split databases

In years gone by, many installations encountered the old limitations of 99 sets or 255 items. In order to accomodate the applications they had designed, the solution was simple: split the database into two or more databases. A similar solution was often used if an installation felt that it was being hurt by the single-threading issue. Those constraints have now been removed, and another interesting problem has replaced them. Those applications which have split database may now consume much more resources than they save (because of the extra DBU's). If you have such a application you may want to try merging the databases back together. Here's how:

1. Choose one database as the primary database
2. Use DBUNLOAD to unload the primary database
3. Add the data elements from the secondary bases to the primary database. Be careful to add the secondary sets and items to the end of the item lists and set lists.
4. Recompile and DBLOAD the primary database
5. Use DB2DISK from the CSL to unload the secondary database set by set [14]
6. Use DISK2DB to load the information back into the primary database [15]
7. Establish an account-wide (or system-wide) user defined command which redirects programs from the old secondary database to the new database

EXAMPLE:

```
DB
OPTION LOGON
FILE DB2=DBPRIME
FILE DB3=DBPRIME
FILE DB4=DBPRIME
****
```

Buffer sizes

After you have determined the minimum number of buffers, you may determine the optimal buffer size. The space available for buffers is now equal to the maximum size XDS allowed (approximately 32K words). The DBB requires about 4000 words for overhead space, leaving 28K words for buffers. The buffer space used by TurboIMAGE is equal to the maximum blocksize allowed plus thirteen words of overhead. That is:

$$\text{buffer\_plus\_OH} = \frac{28K \text{ words}}{\text{buffer\_count}}$$

BLOCKMAX = buffer_plus_OH - 17 ; round up to even number

Block Factor

Great! Now we know what size buffer we want, but guess what? Just specifying a large block size in the schema does not automatically mean that TurboIMAGE will use it! Always check the summary generated at the end of a DBSCHEMA compile. If the value of BLOCKMAX minus the BLK_LGTH is greater than MED_REC, you are wasting space. You may force the SCHEMA compiler to use the extra space by changing the blocking factor (BLK_FAC). For example:

```
Name:     TEST-SET, Automatic
Entry:    KEY-VALUE(1);
Capacity:311 (BLK_FAC);
```

WARNING: Choosing to enlarge the buffers in order to increase the blocking factor, without first calculating the minimum number of buffers is wrong! I have a friend who did not calculate the minimum number of buffers but instead increased the blocksize to 4K words. The result was significantly slower performance. Choosing buffer size and count using the old IMAGE-B rules WILL HURT YOUR PERFORMANCE.

TurboIMAGE and the File System

The improvements HP made did not stop with the restructuring of the control blocks, buffer management algorithms or the obvious visual changes. The interface between IMAGE and the file system has been also tuned up. This is significant. A major enhancement made to MPE V several years ago is called the GLOBAL AFT. Until TurboIMAGE, very little use was made of GLOBAL AFTs. The purpose is to allow a process to request that a file-open (FOPEN) be made globally. Once the file has been opened, any program can read or write to the file without opening it.

Seems straight forward, right? Well it is, sort of... You see, what can now occur is that a program which accesses the database first can open up all the datasets, place them in a globally-opened file list, do its work then close the database. If another program starts to access the database while the first one is running, the

datasets opened by the original program are left open. The second program is responsible for closing the files! What's gained by all of this? Lots! File open and close operations are some of the most resource-consuming operations available on the HP3000. The number that must be performed has now been significantly reduced. A second, equally important, feature of global AFTs is that the file open information is no longer stored in every program's stack but in a special XDS reserved for it. This eliminates the file system error 74 (INSUFFICIENT STACK SPACE) abort [16] which occasionally occurred, leaving the database in a corrupted state! The third use of the global AFT is that HP can now implement a faster I/O algorithm using NO-WAIT I/O.

Are there drawbacks? Yes. With IMAGE-B, a DBCLOSE mode 2 would release the resources associated with the dataset, including closing the file. This no longer occurs! The most significant impact is that if a modify program opens the database first, then several read-only programs open the database, when the modify program ends, the database will still have the files opened for read/write access until the last program (read-only included) has closed the database.

NOWAIT I/O

NOWAIT I/O is not a new feature of MPE, although it is new for HP to use it with IMAGE! NOWAIT I/O is a function of both the hardware and the MPE operating system. Perhaps an analogy would explain it best. Let's presume that we have a Very Important Person (VIP) who will arrive at the local airport and wants to visit the mayor. The trip requires that we travel through a minimum of 20 intersections. The standard WAIT I/O scheme would have several escort cars block each intersection when the light changes, then let the VIP through. Once the VIP is on the other side of the intersection, he waits for the escort cars to get back in front of him so they can block the next intersection. Although this method has the least impact on the local community, it takes considerably more time to travel the distance.

On the other hand, had the method been NOWAIT I/O, the path would have been mapped out ahead of time, with several escort cars at each intersection. Each escort driver knows the approximate time the VIP will arrive at his intersection and automatically closes off the intersection just prior to the VIP's arrival. The result: the car carrying the VIP need never slow down. The NOWAIT I/O scheme requires more resources (in the case of TurboIMAGE, CPU and extra disc controllers).

The use of NOWAIT I/O has another side benefit. Since the route is planned ahead of time and takes less time to travel, there is less time for a terrorist attack (system failure).

If AUTODEFER is not turned on, then TurboIMAGE will ensure that all buffers have been posted before returning to the user. (A read/write to cache is considered complete!) NOWAIT I/O is used only for DBDELETE and DBPUT. These are two intrinsics where multiple I/O's are routine. Once the last I/O has been requested, TurboIMAGE then enters a loop where it will wait until the I/O has been completed (MPE will set a flag as each I/O completes). When the last I/O is done, the DBB is released.

With AUTODEFER, the buffers are not automatically posted. Instead, we retain the changed buffers in the DBB for as long as possible with the hope that more changes can be made to the current set of blocks before we need to start the I/O. When we need to write a buffer, the I/O is started; however, we do not need to wait to see if it has completed. Instead, since we have global AFTs, we will let the next process that requests a buffer perform that check. If the I/O is still pending, this other process must wait!

The non-AUTODEFER I/O should still be as reliable as it was in IMAGE-B; however, as you may have already guessed, AUTODEFER sounds more complicated and risky. It can be especially hazardous to your health if you do not use some form of transaction logging and rollforward recovery. The performance gains, though, are phenomenal and well worth considering!

RECOVERY OPTIONS

The old standby recovery method, ROLLFORWARD, is still in TurboIMAGE. What HP did was improve ILR and add the much-desired ROLLBACK recovery. With ROLLBACK, you no longer purge the database, mount the log tape and then reenact the transactions. Instead, the log tape is mounted and scanned for "recovery blocks". Any incomplete transaction in the recovery block is "backed out". The result is that you may be up and running in minutes instead of hours or days! There is a major disadvantage, though. In order to ensure that all the transaction log records make it to the logfile, user logging uses the serial write queue and does not buffer the transactions in its internal memory buffer. This means that there is at least one additional I/O for every DBUPDATE, DBPUT, DBDELETE, DBBEGIN or DBEND. Furthermore, ILR must also be enabled. The TurboIMAGE manual contains a reasonably complete writeup about the features; be sure to read it. In addition, Peter Kane of HP has written a paper [17] which you should also read.

Caution should be exercised with the new :CHANGELOG feature of MPE. I have received a number of calls from users of a Contributed Library program that I wrote, LOGLIST [18]. It appears that what has happened at several sites is that rather than stopping the logging cycle when backups were done, a :CHANGELOG command was used instead. Then, after the backup was completed, the old logfile was purged and transaction logging was resumed. Later, the users wanted to look at the logfile, and LOGLIST asked for a file that existed months ago, but had long since been purged. The moral of this story: :CHANGELOG SHOULD NOT BE USED TO START/STOP A LOGGING CYCLE! :CHANGELOG IS A TOOL TO BE USED WHILE THE LOGGING CYCLE IS IN PROGRESS. All those files are part of a "log set", which is required to RECOVER or AUDIT the database! [19]

Fair weather database management with Turbo

When is safe safe enough? Next to designing the database, probably the most difficult job for the database administrator is assuring that the information contained therein is correct, logical and syntactically correct (no broken chains). One method to meet the desired goal is to use the full rollback option. This ensures database structure integrity, while also providing the audit trail required to locate and correct semantic errors. If your system is already heavily loaded, this may be unacceptable. The other solution is to use AUTODEFER with rollforward recovery. This ensures database integrity and provides an audit trail, while also providing a boost in performance. The drawback to this is that when a system interruption (sustained power loss, system failure or dead lock) occurs, a slow rollforward recovery must be performed. The "fair weather database management" method says we will resort to enabling rollback recovery at times when we are at a higher risk (thunderstorms, the day after PMs, etc.) and switch to the AUTODEFER/LOGGING option when the sun is out.

Mechanic's notes

One of the enhancements included in TurboIMAGE was a major change in the way that ILR performs its duty. With the previous version of ILR, there was an assurance of a minimum of three I/O's (more if there were an insufficient number of internal buffers allocated). Turbo's version of ILR has reduced this to two I/O's. This was accomplished by simplifying the method of marking a transaction in progress. In the ideal case, the required buffers are modified and moved over to the ILCB. A copy of the transaction logging information to be sent to the user logging interface (via WRITELOG) is placed in the ILCB. This log record is renamed to "Action log" and is used to ensure that the transaction log and ILR file are in sync, so that a rollback recovery can be made. A start timestamp

is added, then the whole ILCB is written out in one I/O (this assumes best case). After the ILR file has been written, the individual buffers are posted to the appropriate datasets. The last step is to signal the transaction complete by writing a new end timestamp in the trailer area of the ILR file.

If there is a system failure, the ending timestamp will be less than the start timestamp. A crash is detected by DBOPEN. Special "hooks" placed in DBOPEN allow it to bring the buffers back into the DBB and post the buffers back to the datasets. The final ILR step is to repeat the command as recorded in the "action log".

One of the interesting observations I made was that the ILR file always appeared to be zeroed out when nobody was accessing the database. Apparently DBCLOSE now cleans up the ILR file, which in turn, makes the process of opening up the database easier.

Finally, here's one other tip. Each record in the ILR file is large enough to hold just one buffer. TurboIMAGE apparently calculates the minimum number of records required to hold the worst case DBPUT or DBDELETE. It then adds additional space so it can hold some global information, the "action log" and the trailer (ending timestamp is here). What all this means is we now have a simple method to find out the minimum number of buffers that should be specified to DBUTIL. Here's how:

1. Turn on ILR
2. :LISTF dbname00,2
3. use the number of records under "EOF" is approximately equal to the minimum number of buffers you should have
4. Use DBUTIL to change buffspecs
   :RUN DBUTIL.PUB.SYS
   >SET dbname BUFFSPECS=eof(1/yourchoice)
5. Optionally Turn off ILR

My wish list

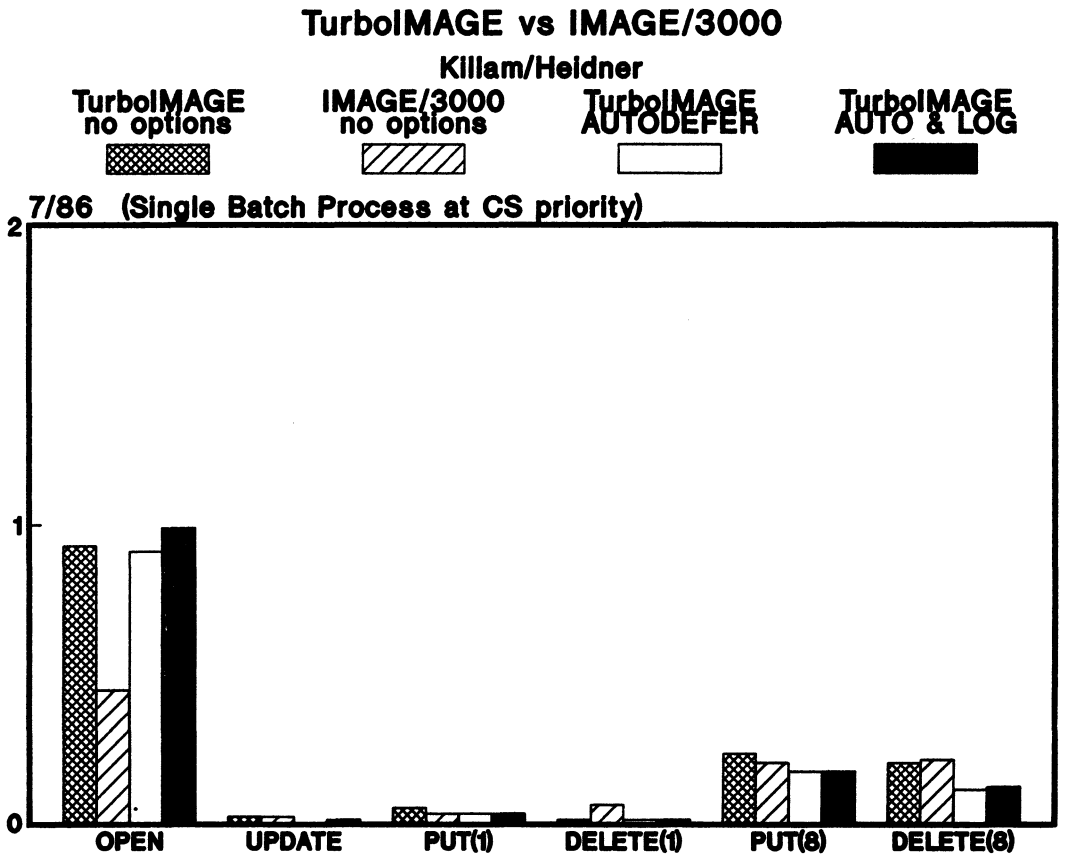
Anytime a do-it-yourself mechanic gets around a new car, there are a lots of OOOOHs and AAAHs. There are also a lot of "what ifs", and "If I had made the car" or "If I had designed the widget!" Well, I have a couple of pet peeves that I wish were fixed. The first is the ability to update critical search items. This one feature would actually increase the versatility of TurboIMAGE and provide added performance benefits to most HP3000 users. The second is a way to "break" database deadlocks without having to resort to a system shutdown and restart. Ideally, this "break" routine would let the Database Administrator (DBA) force the deadlock free without shutting down the database several times.

One simple method would be to simply let the DBA use DBUTIL to mark the DBG or DBB as being corrupted. This would, of course, require that the deadlocked database be recovered, either with DBRECOV or DBUNLOAD/DBLOAD. The advantage would be that only one database would be involved and in a more controlled manner than a system shutdown/restart. (Remember that, with a shutdown, you would have had to recover anyway!) The third wish list item is additional DBINFO calls. I would like to see HP do the same thing for the DBA and TurboIMAGE as they did for the MPE command user. Every piece of information that is available from the DBUTIL SHOW command or from the PROFILER hooks should be available through DBINFO calls. Imagine the tools that could be written, DBOO (like SOO but for databases), monitoring tools that automatically warn you when you need to tune the database, tools that recommend and correct improper buffspecs, etc. Oh well, I always thought the Edsel was a neat car....

FINI

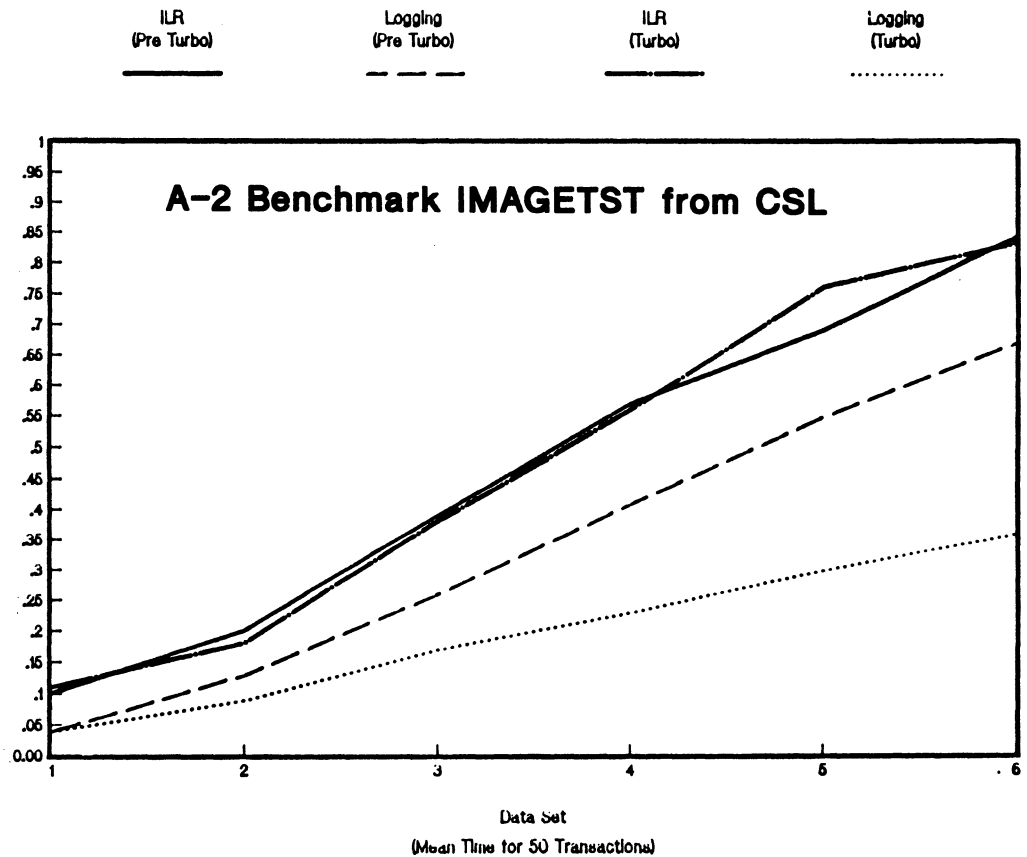
What's the bottom line? Would I recommend that you "buy into" TurboIMAGE? Yes! TurboIMAGE will have been in the field at least two years by the time this paper has been published. Most of the little bugs that invariably show up with new products should be fixed. The new features are very worth asking for. It's true that TurboIMAGE can be a resource hog. However, if you're willing to play by the new rules that TurboIMAGE brings with it, most, if not all, of the performance problems can be minimized or eliminated.

The old rules of thumb for IMAGE/3000 are not necessarily the correct rules of thumb for TurboIMAGE. Unfortunately, the old rules of thumb were the result of many man-years of experience by the established HP3000 community. It may be several more years before the new rules of thumb for TurboIMAGE are well known. You can help! Let's see some papers written about how well PROFILER/3000 works and compares to the standard CSL tools like DBLOADNG and DBSTAT2 for finding problems. More user-developed benchmarks (as evil as they are) would be useful. We need some users to push TurboIMAGE to its limits. What really happens when you specify 1023 items and 199 sets? See all you mechanics next year!



A-1 Benchmark IMAGETST from CSL

DBPUTS


ILR (Pre Turbo)
Logging (Pre Turbo)
ILR (Turbo)
Logging (Turbo)
A-2 Benchmark IMAGETST from CSL
1
.95
.9
.85
.8
.75
.7
.65
.6
.55
.5
.45
.4
.35
.3
.25
.2
.15
.1
.05
0.00
1
2
3
4
5
6
Data Set
(Mean Time for 50 Transactions)

```
ADPAN - Rev 2.10-86195  (C) 1985 The Boeing Co      SAT, JUL 19, 1986,  9:46 PM
DUMP: D1861859.PUB.GOODIDEA            PROGRAM: QUERY.PUB.SYS

   Q   L   SEGMENT NAME    PROCEDURE NAME   P'REL          STATUS
-------------------------------------------------------------------------------
 52575 S  $CACHESEG         CDT'ATTACHIO     00267 PM,XIN,L,CCE
 52560 S  $HARDRES          ATTACHIO         00247 PM,XIN,L,CCG
 52526 S  $FILESYS1A        IOMOVE           00654 PM,XIN,L,CCL
 52333 S  $FILESYS1A        FREADDIR         00232 PM,XIN,L,CCG
 52213 S  $USER              HIDDEN'PT'001    00034 PM,XIN,L,CCG
 52202 S  $USER              HIDDEN'PT'020    77653 PM,XIN,L,CCE
 52142 S  $USER              HIDDEN'PT'012    00050 PM,XIN,L,CCE
 50625 G  $USER             GENMESSAGE       00077 PM,XIN,L,CCE
-------------------------------------------------------------------------------
 50373 G  !TIMAGE11          TRACEMSG         00376 PM,XIN,L,CARRY,CCE
 50212 G  !TIMAGE11          TRACEBLOCKREF    00071 PM,XIN,L,CARRY,CCG
 50115 G  !TIMAGE02          ACTIVATEBLOCK    00467 PM,XIN,L,CCL
 50070 G  !TIMAGE02          GETBLOCK         00046 PM,XIN,L,CCL
 50051 G  !TIMAGE07          DBDELETE         74467 PM,XIN,L,CCL
 47577     SEG'              UPDATE           06700 UM,XIN,TRAPS,L,CCG
 27475     QS08              DBIF             01143 UM,XIN,TRAPS,L,CARRY,CCG
 27220     QS08              SEARCH           00140 UM,XIN,TRAPS,L,CCE
 27214 G  $MORGUE'ABORT      TERMINATE'             PM,XIN,L,CCG
```

TRACE Hooks

New Procedures

A-3 (Turbo)

```
ADPAN - Rev 2.10-86195  (C) 1985 The Boeing Co      SAT, JUL 19, 1986,  9:51 PM
DUMP: D1861860.PUB.GOODIDEA            PROGRAM: QUERY.PUB.SYS

   Q   L   SEGMENT NAME    PROCEDURE NAME   P'REL          STATUS
-------------------------------------------------------------------------------
 52613 S  $CACHESEG         CDT'ATTACHIO     00267 PM,XIN,L,CCE
 52576 S  $HARDRES          ATTACHIO         00247 PM,XIN,L,CCG
 52544 S  $FILESYS1A        IOMOVE           00654 PM,XIN,L,CCL
 52351 S  $FILESYS1A        FREADDIR         00232 PM,XIN,L,CCG
 52231 S  $USER              HIDDEN'PT'001    00034 PM,XIN,L,CARRY,CCG
 52220 S  $USER              HIDDEN'PT'020    77653 PM,XIN,L,CARRY,CCE
 52160 S  $USER              HIDDEN'PT'012    00050 PM,XIN,L,CCE
 50672 G  $USER             GENMESSAGE       00077 PM,XIN,L,CCE
-------------------------------------------------------------------------------
 50440 G  !TIMAGE11          TRACEWRITE       00321 PM,XIN,L,CCE
 50310 G  !TIMAGE11          TRACEBEGIN       00313 PM,XIN,L,CCG
 50144 G  !TIMAGE02          FLUSH'ILR'XDS    00021 PM,XIN,L,CARRY,CCG
 50117 G  !TIMAGE02          LOG'BLOCK'TO'DI  00020 PM,XIN,L,CARRY,CCL
 50100 G  !TIMAGE02          ILR'FLUSH        00041 PM,XIN,L,CARRY,CCL
 50061 G  !TIMAGE02          REQUESTWRITE     00470 PM,XIN,L,CCL
 50037 G  !TIMAGE07          DBDELETE         75273 PM,XIN,L,CCL
 47577     SEG'              UPDATE           06700 UM,XIN,TRAPS,L,CCG
 27475     QS08              DBIF             01143 UM,XIN,TRAPS,L,CARRY,CCG
```

MPE FILE SYS

TRACE HOOKS

ILR

New Procedure

A-4 (Turbo)

ADPAN - Rev 2.01-85252 (C) 1985 The Boeing Co TUE, MAR 18, 1986, 6:45 PM
DUMP: D0771843.DATA, PROGRAM: DALYTST.PROG.

Q	L	SEGMENT NAME	PROCEDURE NAME	P-REL	STATUS
12355	S	$KERNELC	WAIT	00324	PM,L,CCE
12340	S	$HARDRES	WAITFORIO	00271	PM,L,CCE
12317	S	$CACHESEG	CDT'ATTACHIO	00377	PM,XIN,L,CCG
12300	S	$HARDRES	ATTACHIO	00247	PM,XIN,L,CCG
12246	S	$FILESYS1A	IOMOVE	00654	PM,XIN,L,CCL
12053	S	$FILESYS1A	FREADDIR	00233	PM,XIN,L,CCG
11733	S	!IMAGE04	DBGET	76542	PM,XIN,L,CCG
11606		DATABASECALLS	DBUSER	03360	UM,XIN,TRAPS,L,CCG
11231		MAIN	ADDMTECMODEL	01725	UM,XIN,TRAPS,L,CCG
10550		MAIN	DALYMTEC	00411	UM,XIN,TRAPS,L,CCG
01243	S	$MORGUE	TERMINATE		PM,XIN,L,CCG

Note!

DBGET DIRECTLY Calls FREADDIR!

A-5 (IMAGE/3000)

```
ADPAN - Rev 2.10-86195  (C) 1985 The Boeing Co      SAT, JUL 19, 1986,  9:46 PM
DUMP: D1861859.PUB.GOODIDEA          PROGRAM: QUERY.PUB.SYS

F#          FILENAME                FOPT%  AOPT%  RECSIZE RECPT LOGCOUNT
  1 $STDIN                          002244 001700    -80     -1      -1
  2 $STDLIST                        002614 001701    -81     -1      -1
  3 QSOUT                           000614 001401    -81     -1      -1
  4 QSIN                            000254 001400    -80     -1      -1
  6 QSKIB                           002004 000524   -256      0       0
  7 QSMSGCAT                        000005 000420    -80    208      48
  8 TRACEOUT                        000005 002303    -80     -1      -1
  9 IMAGECAT                        000005 000420    -80    192   66976
 10 SAMPLE  <-- Rootfile            000001 000764   -256      6       7
 11 QSSELECT                        002000 000524   -256      1       6
```

NO DATASET FILES

A-6 (Turbo)

```
ADPAN - Rev 2.01-85252  (C) 1985 The Boeing Co      TUE, MAR 18, 1986,  6:45 PM
DUMP: D0771843.DATA.                 PROGRAM: DALYTST.PROG.

F#          FILENAME                FOPT%  AOPT%  RECSIZE RECPT LOGCOUNT
  1 $STDIN                          000305 001300   -1024     -1      -1
  2 $STDLIST                        000705 001301   -1024     -1      -1
  3 $STDINX                         000305 001300   -1024     -1      -1
  4 $STDLIST                        000705 001301   -1024     -1      -1
  5 CATALOG                         000005 000420    -80     256     144
  6 TE  11  <-- rootfile            002001 000764   -256      44      45
  8 TE  132                         002001 000704   -754      39       9
  9 TE  109                         002001 000704  -1018      87       1
 10 TE  105                         002001 000704   -754      82      13
 11 TE  136                         002001 000704   -974    1406  590,167
 12 TE  118   DATA SETS             002001 000704   -764      91    2171
 13 TE  133                         002001 000704  -1012      31       5
 14 TE  104                         002001 000704  -1012     208      11
 15 TE  131                         002001 000704   -930       0       0
 16 TE  101                         002001 000704   -926    9706       3
 17 TE  128                         002001 000704  -1012       0       0
```

A-7 (IMAGE/3000)

```
*46  SAT, NOV 22, 1986,  2:54 PM    QUERY.PUB.SYS                312859586
*---*---*----*--------------*-----*------------*------------*---------------*
*   *   *    *              *     *   Logical  *   Physical *      Time     *
*Pin*Dst*Nest*  Procedure    *Retrn*------------*------------*---------------*
* # * #.*Levl*  Name/Number  *Code *Reads *Writes*Reads *Writes*  CPU  * Wall  *
*---*---*----*--------------*-----*------------*------------*---------------*
>46      1    ARMTRACE                                        312859828
<46      1    ARMTRACE                                        2       1
S46   Stack space used = 535
P46   Trace word = %100037  Trace Flags = %000037
>46      1    DBOPEN                                          312859885
<46      1    DBOPEN                          24894           10      11
S46   Stack space used = 1823
P46   Data Base ID: 0   Mode: 0  Base Name:          [.        .]
P46       Open Cnt: 0    Available Buffers: 0
>46      1    DBOPEN                                          312864580
>46  270 2     DBNewSize                                       312865610
<46  270 2     DBNewSize       2
>46  412 2     DBNewSize                                       312865788
<46  412 2     DBNewSize       2
>46  362 2     DBNewSize                                       312866343
<46  362 2     DBNewSize       2
>46  270 2     SetBufferCount                                  312866530
<46  270 2     SetBufferCount
>46  270 2     SetBufferCount                                  312866703
>46  270 3     DBNewSize                                       312866788
<46  270 3     DBNewSize       2
<46  270 2     SetBufferCount -1
<46      1    DBOPEN                 30     1      30     1      1611    2516
S46   Stack space used = 1758
P46   Data Base ID: 242   Mode: 1  Base Name: TE      [DATA    .        ]
P46       Open Cnt: 1    Available Buffers: 34
>46      1    DBINFO                                          312867494
<46      1    DBINFO                                          2       2
S46   Stack space used = 1704
P46   Data Base ID: 242   Mode: 501
P46   Qualifier: 0   Buffer Length Returned: 1
>46      1    DBINFO                                          312870988
<46      1    DBINFO                                          5       4
S46   Stack space used = 1704
P46   Data Base ID: 242   Mode: 203
P46   Qualifier: 0   Buffer Length Returned: 60
>46      1    DBINFO                                          312871479
<46      1    DBINFO                                          2       2
S46   Stack space used = 1704
P46   Data Base ID: 242   Mode: 901
P46   Qualifier: 0   Buffer Length Returned: 1
>46      1    DBINFO                                          312871977
>46  270 2     DBOpenDset                                      312872101
<46  270 2     DBOpenDset
<46      1    DBINFO                  1             1         328     387
S46   Stack space used = 1803
P46   Data Base ID: 242   Mode: 202
P46   Qualifier: 1   Buffer Length Returned: 17
>46      1    DBINFO                                          312872838
<46      1    DBINFO                                          4       4
S46   Stack space used = 1704
P46   Data Base ID: 242   Mode: 104
P46   Qualifier: 1   Buffer Length Returned: 2
>46      1    DBCLOSE                                         312873291
```

A-8 (Trace file)

```
<146     1    DBINFO                                                  3        2
S146  Stack space used = 1704
P146  Data Base ID: 130   Mode: 102
P146  Qualifier: 1   Buffer Length Returned: 13
>146     1    DBINFO                                                  159343597
<146     1    DBINFO                                                  3        3
S146  Stack space used = 1704
P146  Data Base ID: 130   Mode: 102
P146  Qualifier: 2   Buffer Length Returned: 13
>146     1    DBINFO                                                  159344853
<146     1    DBINFO                                                  3        3
S146  Stack space used = 1704
P146  Data Base ID: 130   Mode: 102
P146  Qualifier: 3   Buffer Length Returned: 13
>146     1    DBINFO                                                  159346235
<146     1    DBINFO                                                  3        3
S146  Stack space used = 1704
P146  Data Base ID: 130   Mode: 102
P146  Qualifier: 4   Buffer Length Returned: 13
>146     1    DBLOCK                                                  159347568
<146     1    DBLOCK                                                  4        4
S146  Stack space used = 1692
P146  Data Base ID: 130     Mode: 6
P146  Lock Wait Flag: FALSE    Lock Acquire Time: 0
>146     1    DBPUT                                                   159347967
>146 410 2     GetBlock                                                159348080
>146 410 3     ActivateBlock                                           159348188
R146      Referenced block 0 from set 1.
<146 410 3     ActivateBlock  231
<146 410 2     GetBlock       231
>146 410 2     ReleaseBlock                                            159348553
<146 410 2     ReleaseBlock
>146 410 2     GetBlock                                                159348760
>146 410 3     ActivateBlock                                           159348866
>146 410 4     RequestRead                                             159348953
<146 410 4     RequestRead
R146      Referenced block 1 from set 2.
<146 410 3     ActivateBlock  2255
<146 410 2     GetBlock       2255
>146 410 2     ReleaseBlock                                            159349419
<146 410 2     ReleaseBlock
>146 410 2     GetBlock                                                159349626
>146 410 3     ActivateBlock                                           159349732
R146      Referenced block 1 from set 3.
<146 410 3     ActivateBlock  1243
<146 410 2     GetBlock       1243
>146 410 2     GetWriteAccess                                          159350116
<146 410 2     GetWriteAccess
>146 410 2     GetBlock                                                159350336
>146 410 3     ActivateBlock                                           159350442
R146      Referenced block 1 from set 2.
<146 410 3     ActivateBlock  2255
<146 410 2     GetBlock       2255
>146 410 2     GetWriteAccess                                          159350785
<146 410 2     GetWriteAccess
>146 410 2     GetBlock                                                159350998
>146 410 3     ActivateBlock                                           159351106
R146      Referenced block 0 from set 3.
<146 410 3     ActivateBlock  1749
<146 410 2     GetBlock       1749
```

need block 1, set 1
MUST READ

need block 1, set 2
ALREADY IN MEMORY!

A-9 (Buffer procedures)

References

[1] TurboIMAGE Data Base Management System reference manual printed 12/85, Chapter 8

[2] IMAGE/3000 Data Base Management System reference manual

[3] Griffin, Doug, Introducing TurboIMAGE, Communicator 3000, Volume 2, Issue 7, page 2-1 (U-MIT)

[4] Griffin, Doug, IMAGE/3000 Changes for MPE V/E and Disc Cache, Communicator 3000, Volume 2, Issue 1, page 5-25

[5] CSL library, INTEREX
680 Almanor Ave.,
Sunnyvale, CA 94160, USA

[6] CSL library

[7] Chen, Doris, "TurboIMAGE Internal file Structure", Proceedings INTEREX HP3000 Madrid Conference, 1986, page 123

[8] Chen, Doris, ibid.

[9] Russell, Marguerite, The IMAGE/3000 Handbook, WORDWARE, 1984, Seattle Washington

[10] TurboIMAGE Data Base Management System reference manual printed 12/85, page 10-5

[11] IMAGE/3000 Data Base Management System reference manual

[12] Griffin, Doug, Introducing TurboIMAGE, Communicator 3000, Volume 2, Issue 7, page 2-1

[13] TurboIMAGE Data Base Management System reference manual, printed 12/85, page 7-8, "Logical Transactions and Locking"

[14] CSL library

[15] CSL library

[16] HP Response Center, STACK OVERFLOWS: Causes and Cures For COBOL II Programs, Document P/N 5958-5824/2649

[17] Kane, Peter, TurboIMAGE Run-Time Options: Balancing Performance with Data Base Integrity", Proceedings of INTEREX HP3000 Madrid Conference, 1986

[18] CSL library

[19] TurboIMAGE Data Base Management System reference manual print 12/85, Chapter 7, pp 7-23, 7-36

Author

Dennis Heidner received his BSEE degree from Montana State University, Bozeman, Montana. Mr. Heidner has written "Transaction Logging and Its Uses", presented at the 1982 HP IUG. He was co-author of two papers, "Transaction Logging Tips" and "IMAGE/3000 Performance Planning and Testing", which were presented at the 1983 HP IUG in Montreal. In 1984 he presented the paper "Disaster Planning and Recovery" at the HP IUG conference in Anaheim. Mr. Heidner is a co-author of The IMAGE/3000 Handbook, published by WordWare, Seattle, Washington. He has written technical articles which have been published in several magazines.

Mr. Heidner is a member of the Association for Computing Machinery since 1982 and a member of the Institute for Electrical and Electronic Engineers (IEEE) since 1974. He is a member of the IEEE Computer Society.

ASTAR is Born!

Terrell Haines and Dennis Heidner
Boeing Aerospace Company

Abstract

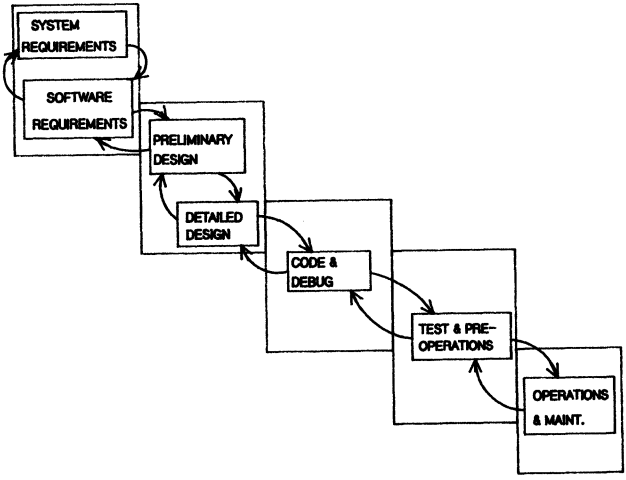
What? Where? When? How? These are four commonly asked questions when the programming staff is asked to make enhancements to existing in-house applications. The usual scenario is to pull out last year's dataflow diagrams, structure charts, flowcharts and source listings, flip a coin and hope that the analysis of the impact will be correct. In order to make this process easier, most programming shops have naming conventions and standard guidelines for coding styles. While these are important, they are still a manual process in an automated environment. What is needed is a programmer's toolbox which will allow the use of a favorite editor but automate the manual functions. This paper covers our search of available third-party tools. After we could not find an acceptable package, an in-house solution (ASTAR) was developed. We will discuss some of the pitfalls, as well as free CSL programs which can be used to implement similar programming environments at your site.

Introduction

Imagine, for a moment, that the company you work for has been awarded a multi-million dollar contract to produce widgets. The management information system, which currently resides on an HP3000, cannot handle the increased demands without modifications to the software. In order to accomplish the change, you request that you have exclusive access to the keypunch, card reader, sorter and card printer/interpreter for at least two months. You will, of course, have a better estimate of the time to make the modifications after you use the sorter and card printer to find all the lines that read from the "STOCK-ON-HAND" dataset.

By now, you must think: "This is insane! Why use old, obsolete technology on an HP3000? After all, one of the most modern and sophisticated programs manages our factory! (I designed it myself!)". Unfortunately, though, it's a simple fact of life that the software development/maintenance groups are expected to use new techniques and methods with tools that were developed ten or fifteen years ago. Even though the president of the company may boast about the new, super-duper, paperless system used in the factory, software designers are expected to use tools which leave them paper-bound. In order to reduce the paper shuffling, special conventions are used to name program modules, source files, datasets, etc. Several librarians are hired and instructed to rigidly enforce the standards, lest chaos break out!

How to maintain control of software associated costs has been the focus of many studies. Various authors have proposed the 90/10 rule, which states that, for any system, 90% of the cost is software and 10% is hardware.[1] In order to understand how software costs are influenced, several "life-cycle" models have been developed. The most popular of these has been attributed to Larry Boehm.[2]



Elements of a Programmer's Toolbox

Let us first review the tools available to most programmers and how they fit into the software design, development and maintenance

cycle. Some of these tools are absolutely required in order to develop software, while others provide productivity improvements for the developers. One observation which should be made is that there is a far smaller selection of software tools for the HP3000 computer than most other manufacturers' machines. Perhaps this has been caused by the tight control HP has maintained on the architectures of the hardware and the MPE operating system. Tools are available from three areas: HP, third party vendors and the INTEREX contributed library (CSL/3000).

Computer-aided environments (CAE)

At the 1986 Structured Development Forum VIII, held in Seattle, Washington, more than thirty vendors showed CAE packages which assist system analysts and programmers in the structured design of applications. Many of these packages were written to execute on personal computers, DEC computers and IBM mainframes. The software packages are intended to provide an environment where software engineers can define, document, check, edit and maintain the software specifications for complex projects. The specifications included Data Flow Diagrams (DFD), structure charts, Warner-Orr charts and more. Unfortunately, there are no such tools native to the HP3000, even though one of the vendors was HP! (Their product is called HP Teamwork/SA.)

Editors

The most-frequently-used software development/maintenance tool on the HP3000 is the editor. HP supplies one editor, called EDIT/3000, as part of the Fundamental Operating System. This editor provides the minimum features required for the software life cycle. HP provides a means to enhance this editor's functions with user-callable procedures but these advanced features are generally not used by most installations because of the training necessary.

HP also offers what they refer to as the Text Document Processor, (TDP). This editor has corrected many of the original shortcomings of EDIT/3000 by including such features as document formatting, an enhanced find command, greater macro capability, better online help, block mode (page) editing and the ability to easily access other programs without exiting the editor. TDP, however, does not support many of the features that EDIT/3000 does. These include user-written procedures and editing variable length files.

Robelle Consulting offers a programmer's editor called QEDIT. This editor has been tuned for speed. By having a high speed editor, you might think you would have to give up functions. QEDIT, though, proves this to be false. QEDIT allows the user to invoke

UDC commands while editing, comes with a "scribe" function, operates in screen mode and adds many features that I wish TDP had. Text files saved by QEDIT have additional QEDIT information stored within the file's user label. Robelle provides special QEDIT access routines which allow user programs to take advantage of the stored QEDIT information.

FSEDIT, from SYDES, is another editor designed with the programmer in mind. FSEDIT offers many of the same features of TDP and QEDIT, in addition to split screen editing (access to multiple files at a time), and a built-in COBOL program generator.

HPTOOLSET, by Hewlett-Packard, is an integrated program development package, especially suited to aid the COBOL programmer. It contains an editor, a direct link to HP3000 COBOL compiler and segmenter, a symbolic debug facility and a workspace/file manager. While the package claims integration, HPTOOLSET, according to the manual, "... is not intended to be compatible with COBOL 68, EDITOR/3000, DEBUG or any other programming tools currently available on the HP 3000." [3]

My favorite from the CSL library is QUAD. QUAD provides high speed editing, version control, and an undo command. Document formatting is accomplished by using GALLEY (also from the CSL library) or the TDP formatter. One of the most unusual features of this powerful editor is that the SPL source code is also in the library.

Compilers

The compilers available for the HP3000 include BASIC, COBOL, FORTRAN, PASCAL, RPG SPL and C (available from third party vendors). A version of FORTH is available from the CSL3000 release A0 tape. Not available are complete implementations of PROLOG, LISP, SNOBOL, PL1 or ADA. Fourth generation packages available include products such as BRW, TRANSACT, RAPID (from HP), POWERHOUSE products from COGNOS, PROTOS from Protos Software Company, FLEXIBLE/3000 from SAGES-AMERICA, SPEEDWARE from InfoCenter, PDQ/Quiz from Tymlabs and FASTRAN from Performance Software Group.

Linkers

The linker-of-choice on the HP3000 is MPE Segmenter, which is also "the only game in town". Seldom is the Segmenter divorced from a compiler and used as a stand-alone package. However, there are instances when the situation calls for analysis of how Segmenter is affecting a job. In that case, the analyst must turn to a very powerful tool, the Segmenter manual. It not only explains the Segmenter, it also introduces the concepts of "virtual memory" and

"segmentation". These two ideas are keys to the operation and use of this tool. It also explains such concepts as Relocatable Binary Modules (RBMs) and User Subprogram Libraries (USLs), using easy-to-grasp analogies.

Segmenter is a dynamic, run-time linker; that is, final links to segmented libraries (SLs) are established when the program is run. This explains why, when a LIB= statement is not included in the RUN statement, MPE sometimes proclaims "UNRESOLVED EXTERNAL REFERENCE" or "PROGRAM LOADED WITH LIB=G". Those final connections to the outside world must be made.

If a program is PREPed to a given USL several times, and no CLEANUSL is invoked to compact the USL, the Segmenter will create several sequential versions of the RBMs. This allows an analyst to actually return to a previous version of a given module. Of course, it also requires the analyst to keep records of what is in each of the various versions. This is perhaps a small cost for the available power.

Debugging Tools

MPE Debug is the Hewlett-Packard tool which is used to locate and correct errors interactively in programs. Invoked at run-time, Debug can set breakpoints which will temporarily stop program execution and turn control over to the analyst. Various commands allow viewing of memory in real-time, during program execution. In addition, Debug gives the ability to actually change the value of memory locations during execution. The STACKDUMP intrinsic enables a selective memory dump to the screen, the printer or other output device.

MPE provides a command, :SETDUMP, which allows the user to request a copy of the stack markers, the user data portion of the stack or both to be printed on $STDLIST whenever a program aborts.

SOOT.PUB.TECH, from the CSL3000 tape, may be used to capture an active program stack for later analysis. The method of doing this is to use the SHOW command to find the PIN number for a given process. Give the DUMP command with the PIN number following, and SOOT will produce a formatted dump file with the name Ddddhhmm, where ddd is the Julian day, hh is the military time hour and mm is the minute at which the dump was created. If multiple dumps are created within the same minute, the mm figure is incremented by one for each new dump file.

ADPAN.PUB.TECH, from the CSL3000 tape, was created by Dennis Heidner to address shortcomings in STACKDUMP. Equipped with an

on-line help facility, it provides visibility of the segment call paths, file information, memory locations relative to the current stack marker and miscellaneous program information for a given formatted dump file.

A companion to ADPAN is SNAPSHOT, an intrinsic which may be called by a running process in order to take a picture of itself. It creates a stack dump, with a name in the same format as SOOT.PUB.TECH.

Crossreference Tools

Crossreference options are available in all of the compilers offered by HP. The option is turned on by a $CONTROL card placed at the start of the source code being compiled. In addition, there are several user-written crossreference tools in the CSL. These include SPLXREF (for SPL), BASXREF and XREFB (for BASIC), COBMAP, COBXREFA and SYSXREF (for COBOL) and RENUMBER (for FORTRAN).

There are a few programs which help tidy up the source code, but very few flowcharting programs are available. This is a sharp contrast from most other minis and microcomputer systems.

Dictionary

HP, COGNOS and several other vendors offer data dictionary products. Dictionary/3000 from HP can be used as a standalone product or used wtih their 4GL products, BRW, RAPID and TRANSACT. COGNOS' dictionary is required by their POWERHOUSE products. COGNOS also provides a utility which allows migration of their dictionary information into HP's dictionary.

Program Generators

On the CSL3000 release B0 tape can be found COBGEN, a COBOL 68 Program Generator. This program assists in the initial stages of writing a COBOL 68 program by prompting the user for necessary entries. Some extra features are support for the QUAD editor from inside COBGEN and support for the COBOL COPY statement. Existing source code can also be modified by use of a parameter in the RUN statement.

FORTRAN preprocessors are available from the CSL3000 tapes, also: TELETRAN, from release 09; RATFOR, from release A0; FTN from releases B0 or ANAHEIM.

Tymlabs offers a product called PDQUIZ which will generate HP3000 object code from COGNOS' QUIZ language. The Protos Software

Company offers a 4GL language, which, in essence, allows your programming staff to write in a pseudocode which then translates into COBOL. Q-GEN, from Proactive Systems Ltd, will generate COBOL report programs from QUERY commands.

Comparision Tools

BLDTEXT, written by Karl Smith, is available in the CSL library. BLDTEXT will take a current version and an old version of a source file and produce a file which contains the $EDIT commands which reflect any new lines, deleted lines or updates made on the new file. The files must both be numbered ASCII files. The output of BLDTEXT is compatible with the HP compiler standard for master and edit files. [4] FILECOMP and COMPARE are two other file comparision routines from the CSL library.

SCONS, by Corporate Computer Systems, compares two files and prints a listing of differences, allowing one to quickly spot specific changes between two revisions. It creates a file of difference records called a "delta" file, which allows reconstruction of previous revisions.

S/COMPARE, by Aldon Computer Group, is a source file comparison program for identifying differences between any two versions of a program.

HARMONIZER, by Aldon Computer Group, is a source comparison program for multiple versions of a file, which can produce a file of merged records of the input files. HARMONIZER supports COBOL, FORTRAN, PASCAL, SPL, COGNOS, TRANSACT, BASIC, COBOL Copy Libraries and any character data files whose record lengths do not exceed 80 bytes. Harmonizer is a spawned process which extends S/COMPARE's capabilities, allowing it to compare up to 16 versions of a file. According to company literature, "an output file can be produced that is a composite of all compared versions of a program, optionally annotated with language-specific comments to describe the source of insertions and deletions."

O/COMPARE, by Aldon Computer Group, compares object code files to verify that program files that are expected to be the same are the same and that a production module was created by the current source file.

Job Control Language (JCL) Aids

PUDC, from the CSL3000 tape, allows the capability of programmable UDCs. These UDCs may include such commands as IF, GO and GOSUB. Streaming of jobs with insertion of parameters is also possible.

MPEX extends the capabilities of MPE so as to allow operations upon entire filesets or even file subsets. A demo of MPEX is on the CSL/3000 Release A0 tape. A fully-supported version is available from VESOFT. MPEX users often become addicted to it power and reportedly have refused to accept jobs at sites without MPEX!

There are a number of packages which allow the queueing of jobs for submittal later at night. The most common ones from the CSL are SLEEPER and JOBQUEUE. There are a number of vendors which offer data center management tools which help schedule and track when jobs should run.

JOB CONTROL SYSTEM/3000, by Diamond Optimum Systems, provides a real-time audit trail for job and session execution. It also provides an expanded :SHOWOUT function. The audit trail information is stored in an IMAGE database and is integrated with DOCUMENTATION/3000.

Documentation Aids

HPSLATE, from Hewlett-Packard, provides a casual word processor designed to be used by business professionals. It features full screen, page-oriented editing and uses the function keys of HP terminals. Due to the question-and-answer dialogue designed for infrequent use, it is not appropriate for other than a casual user. It is available in Italian, French and German.

HPWORD, from Hewlett-Packard, is a shared resource word processor, designed to be used with a special HP Word Processing Station, connected to the HP3000. Graphs and charts may be inserted into documents.

LASTWORD, from Trident Data Systems, operates on HP block mode terminals. It allows full screen editing and has an on-line help facility. There are no embedded commands, although it does allow text enhancements such as underlining and boldface.

SPEEDDOC, from Bradford Business Systems, is a word processing and office automation system. The word processor has on-line help and full-screen editing. MPE commands may be executed from within SPEEDDOC. Data from IMAGE, KSAM and MPE files may be joined directly to word processing documents. Numerous printers are supported. In addition to the word processor, electronic mail, tickler files, room/equipment scheduling, mass mailing facilities and a spelling checker are included in SPEEDDOC.

GALLEY, from the CSL3000 tape, is a batch text file formatter which uses Edit/3000 or TDP files as input. The output is on a printer.

GALLEY is directed by embeddeded commands. There are also versions of the popular UNIX like NROFF document formatter in the CSL.

DOCUMENTATION/3000, by Diamond Optimum Systems, is a program documentation package which includes a wide-ranging on-line cross-reference facility.

Spelling Checkers

SPELL/3000, from Bradford Business Systems, is an interactive spelling checker for standard ASCII files. New words can be added to the dictionary, or separate user dictionaries can be created and modified. A list of most-misspelled words is interrogated first. If the word is found there, it is automatically corrected. Actions are function-key-driven.

Tracking Tools

ROBOT/3000 AUTOMATIC DOCUMENTER, by Productive Software Systems, is an on-line indicator of proposed changes to programs. It produces a display of which programs are affected by a change and the line(s) to be changed.

ROBOT/3000 AUDITOR, by Productive Software Systems, keeps track of file modifications. The amount of history retained is user-definable. It also alerts the user to file purgings, so that an accidental file purge can be restored before the backup file is erased. No manual input is required. ROBOT supports all MPE files.

ARCHIVE/3000, by Fourhills Technology Group, monitors versions of files. It removes older copies of a file from disk to tape and, if desired, will delete the original file. ARCHIVE has a file comparison function. File security functions are available, also.

LITSCAN, from the CSL3000 tape, was originally designed as a bibliographic retrieval system, based on keywords. However, it apparently could be used to track program documentation.

Project Planning

Very few companies can ignore the bottom line, COST. Meeting the goals for a new software system, on time and within budget requires careful project planning and monitoring. The CPM and PERT are the two most popular methods for planning large complicated projects. The CSL library has several contributions which can be used by a development team. These are PERT, SCHED, and PRTCHRT.

The Hewlett-Packard Business Systems Software Solutions manual identifies at least two project planning packages. N5500 Project Management System, from Nichols & Company, is "... an interactive project management system for planning, simulation, tracking & documentation of projects". TASK TRACKER, from Medina Marketing Group, "... tracks and reports on the progress, status, schedules and costs of defined projects and their subordinate tasks".

Desired Features

Thus far, we have covered the common tools used by the design and development team. As you can see, there is a great deal of diversity in the products that are available (we ran out of space). Now imagine for a moment, not the gloomy picture we started the paper with, but a utopia. Every part of the specification and design process has CAE tools to assist you. Even though the code is to run on an HP3000, you can use your favorite PC. When you are forced to make last minute changes in the code, the DFD's and structure charts are automatically updated. When it's time to write the user's documentation, all you have to do is push a button. The release bulletin is all automated. Test scripts are generated which provide the degree of coverage that is specified in the design document. Any errors which later arise (there are not many!) in production will automatically flag the sections of code and/or the specifications which are at fault. Although this might sound like idle daydreaming, it's what has been called "Imagineering" by the Disney design laboratories.

While "imagineered" designs are sometimes only a dream, the dream software environment should become reality. During the past five years, we have seen an explosion of CAE tools which allow the computer hardware engineer to design complex circuits that were impossible to imagine five years ago. These new tools are capable of accepting specification/design statements and generating the internal signal routing, test patterns/vectors, and perform design rule checks.

Let us imagine then that it is within our power to create some form of our ideal. What would it be like? Any "new and improved" programming environment would need to accommodate most, if not all, of these tools. Before starting such an ambitious project, we set down a few objectives which we wanted to meet. They were:

Womb-to-tomb Environment

The ideal programming environment provides a uniform set of tools which can be used from "womb-to-tomb" by directing the output of

one tool into the next tool. It's important to remember in the model presented by Boehm that you may sometimes need to backtrack and adjust work done in the previous step. The ideal tool will accomodate this to the extent that, if a problem in a program would require the use of a different algorithm, the dataflow diagrams and structure charts should be flagged as obsolete or invalid. Enough information should be sent backward to simplify the updating of the design and requirements documents.

Allow Use of Favorite Tools

One of the most significant costs of the software life-cycle is the cost of training and the associated learning curve. The ideal environment must allow the use of the programmer's favorite editors, compilers, etc.

Enhance/augment Programmer's Productivity

The programming environment must improve the productivity of the team. Although this seems obvious, all too often software tools are written which require you to adopt their standard coding conventions and use a specific development methodology. Often these new rules are difficult to use.

Flexibility

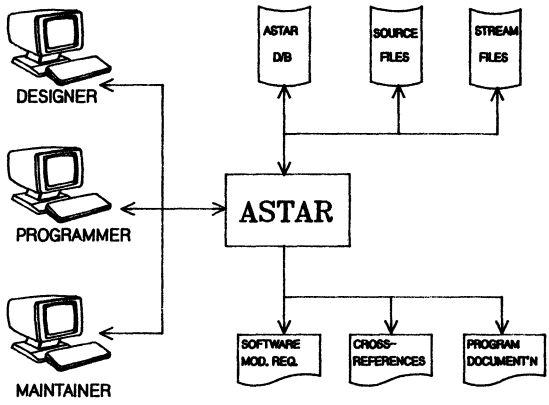
There has been increasing interest in a new software development called prototyping. With this new development process, much of the traditional life-cycle is eliminated. Instead, using a newer fourth- generation language, a software specification is written, compiled and tested by the end-user. If the application does not perform as expected, the specification is modified and recompiled. Prototyping allows the end-user to become involved in writing the specifications, thus eliminating much of the traditional collection of system analysts, programmers and coders.

Open Architecture

No one individual or company can anticapate every potential use for the programming environment, nor can the languages of the future be predicted. For this reason, it is critical that the programming environment be an open architecture. As the need arises, we must be able to write new tools which are easily dovetailed into the current collection of tools. The success of such an open scheme may be clearly seen in the diverse collection of MSDOS and UNIX tools.

ASTAR

ASTAR (Automated Software Tracking And Reporting) was our answer to the shortage of programmer tools for the HP3000. Developed in-house, it provides an integrated source for producing programmer information. Where a specific tool did not exist, it was written. If a tool did exist, it was dovetailed into the system. ASTAR was written to be expandable. New functions, if not directly attachable to the system, may be added to the main menu.



ASTAR FLOW DIAGRAM

ASTAR provides an automated means to track software modification requests, work progress, software changes and also estimate the impact of changes that modifications to one module might have on others. ASTAR, as such, provides the framework for a programmer's workbench on the HP3000 computer.

ASTAR was designed so that it can be used with an application which has already been designed and is in use. Special stream jobs were created which crossreference and index all files withing the application, once this is done, only the files which are modified are reexamined.

ASTAR Programs

ADEDFILE is a program which scans the ASTAR database and removes all references to files and datasets that no longer exist. The

algorithm used by ADEDFILE is quite simple. We just serially read the master set which contains the file names, then we check to see if the file still exists (use programatic :LISTF). If it is no longer, then the references to the file are removed from the ASTAR database.

AMERGE is a general purpose INCLUDE/COPYLIB program, which also supports the $EDIT control options. AMERGE is intended to be the "shell" through which current vendor-supplied programs must work. AMERGE is table-driven, with the ability for the users to add "new" packages by describing the interface. This description would include whether or not we are to direct the processed file to a "pipe" or write to a file. AMERGE has process-handling capability so it can create and activate the desired program.

APRINT is a command-driven program which will locate and print the section of any text file containing a specified object. APRINT opens up the ASTAR database, then parses the command passed to it. The output of APRINT can be directed to any file. APRINT will automatically set up a file equation "AFILE" which points to the last file it was printing. This allows the user to locate and view a file with APRINT, then recall it into the editor without ever knowing the name of the source file!

ADPAN is a program which assists in the maintenance of programs which have been released into production. ADPAN (and SNAPSHOT) work together to capture program aborts. This post-mortem information significantly reduces the time required to isolate program aborts.

ASNAP is a program which prints a summary for snapshot files. The summary includes the program name, date and time of abort.

ATREE is a program which will generate a procedure tree, starting at a name specified by the user. ATREE takes the procedure name specified by the user, then looks up all procedures called from the one specified. For each procedure it finds, ATREE in turn recursively calls itself in order to process the next lower level. The output can be redirected to another file, where it can be processed into a structure chart.

DSETXREF is a program which reads a specfied source file and locates all references to datasets in a specified database. DSETXREF accepts a specific database name, then builds a table of datasets. Then every text file that has been modified is scanned looking for the dataset name. As a name is found, it is added to the index.

ERRXREF will scan all source files and update the index indicating where a specific error number is used. There are no standards on how an error number must be coded. This presented a special problem when we were developing ASTAR. The final solution was to allow "trigger" and "terminate" strings to be specified. Any character between the trigger and termination will be considered an "error number". This implementation allows "error numbers" to consist of any ASCII character, for example: "CIERROR 976".

JOBXREF will scan all stream jobs and build an index indicating where and how files are accessed. JOBXREF looks at the first line in the file and verifies that we are working with a MPE JOB command. Then the "stream" character, (typically :, ! or #), is retained. Any line in the file which begins with this stream character will be considered to be a command and not data. JOBXREF looks at the logon name, then reads into a table the user defined commands (UDC) for that specific user. As a command in the stream file is encountered, the UDC table is scanned. Any file which is referenced by the command is added to the index. JOBXREF has been written to understand the syntax for the software tools supplied as part of FOS. This includes QUERY and FCOPY.

PROCXREF will scan the program source code and build an index which contains characteristics of the procedures and what procedures are referenced. PROCXREF is syntax-driven, enabling it to determine characteristics of the procedure it is analyzing. Every time a new procedure is encountered while scanning source files, the summary compiled to that point is added into the database.

PROGXREF scans the object files for revision numbers and picks up the program characteristics.

UDCXREF reads all the udc commands in use for the account and builds an index of who uses which commands and what they are. Later JOBXREF uses this UDC index to assist in tracking where, when and how files are referenced.

How ASTAR Detects File Changes.

The ASTAR stream jobs use DIRK from the TECH account to locate which files have been modified in the last 24 hours. The fully-qualified name of these files are written to a workfile. Later, the ASTAR programs use this workfile, which concentrates the indexing on just the modified files. There are several other programs in the CSL which could be used to perform the same task, or MPEX could be used. Another method would be to require the system administrator to turn on logging of FCLOSE records; then a special

program could be written to extract these records and build the worklist.

ASTAR Screens

ASTAR is menu-driven, with screens written in QUICK from COGNOS. An example of the opening ASTAR screen is shown in Appendix A-1.

ASTAR SR'S

The SR screen is used to enter an inquiry on software modification requests. An example of this screen is shown in Appendix A-2.

The fields are defined as follows:

SR# - Modification request number (automatically assigned when entering requests, although you may override)

NAME - Program, job or procedure name where problem most likely occurs.

REVISION - The projected revision number that the software will have at the time this SR is closed

TOPIC - Is the SR to correct a bug or make an enhancement? Allowable values are:

* Perfective - enhancements of the package
* Adaptive - changing to keep up with new rules or operating procedures
* Corrective - correcting a mistake

USER - The logon ID or group of users requesting the change

REQUESTOR - The name of the individual requesting the change

ASSIGNED TO - The analyst to be responsible for planning, evaluating, designing and implementing the change

DATE SUBMITTED - The date the SR was logged into ASTAR.

ASTAR JOB INFO

The JOB INFO screen provides a summary report on the function of a stream job, the jobname, its purpose, author, revision and when it was last modified. An example of the screen is shown in Appendix A-3.

The fields are defined as:

JOB SOURCE - The stream job's source file

JOBNAME - The job/session id, (!JOB jobname,<<user.acct>>)

ACCOUNT - The account in which the stream job resides

GROUP - The file group in which the job resides

FIRST REC. - The first record number of the stream job. This feature is used by sites in which all

stream jobs are in one large file, and a program extracts jobs to be launched

LAST RECORD - The last record in the stream job

REVISION - The version number for the stream job

TYPE - The type of stream job; for example, report, compile, maintenance, system operation, etc.

ASTAR DATE - The date the job was last modified

PURPOSE - The purpose of the stream job.

JOB/FILE INFO

The JOB/FILE INFO screen provides a easy method for the programmer or analyst to determine where and how files are used. This screen provides only for the viewing of information, no changes may be made. An example of this screen is shown in Appendix A-4.

The fields are defined as:

SOURCE - The file to be checked

UDC COMMAND - If there is anything here, the file was referenced through a UDC command in a stream job

MPE COMMAND - The MPE command invoked when the file is referenced

STREAM JOB - The stream job which contains the reference

R# - Logical record number in the stream job where the SOURCE file was referenced

DATE - The date the file was crossreferenced.

PROGRAM INFO

The PROGRAM INFO screen provides a quick summary of programs, purpose, revision history, comments and size information. An example of this screen will be found in Appendix A-5.

The fields are defined as follows:

NAME - The internal procedure name for the program to be checked

SOURCE - The executable object file for the program

REVISION - Revision number (if any) for the program

ASTAR DATE - The date the program was crossreferenced

PURPOSE - The purpose of the program

COMMENTS - Comments about the program

MAXDATA - The maximum stack size with which the program was PREPed

SEGMENTS - The number of segments in the program file

CODE SIZE - The size of the object file

COMP. JOB - The stream job which will recompile the program

PROCEDURE INFO

This procedure provides a general summary about the construction of a piece of particular program procedure. An example of this screen is shown in Appendix A-6.

The fields are defined as follows:

NAME - The procedure name

SOURCE - The editor source file for the procedure

PURPOSE - What this procedure is supposed to do.

INITIALS - The initials of the author for the procedure

TYPE - The type of procedure, SUBroutine or FUNction.

LANGUAGE - The high level language in which the procedure is written

STATUS - The current condition of the procedure:

- o DONE = in production
- o PLAN = planned
- o CODE = being coded
- o DBUG = being debugged
- o MAIN = on-going maintenance is being performed

REVISION - The revision number of the procedure

ASTAR DATE - The date the procedure was crossreferenced

FIRST REC. - The logical record number of the first line of the procedure in the source file.

LAST RECORD - The last logical record number of the procedure

LINES OF CODE - The number of lines of executable code (comments and data declarations have been excluded)

LOOP COUNT - The number of DO loops, or DO UNTIL-type statements in the procedure

OF GOTO'S - The number of GOTO's in the procedure

OF IF'S - The number of conditional branches in the procedure

EXPRESSIONS - The number of assignment expressions in the procedure

OF I/O'S - The number of I/O-type statements which are used in the procedure (i.e. WRITE, READ, DISPLAY)

CALLS - The number of references to other procedures from from within this procedure.

FORMATS - The number of FORTRAN FORMAT statements

OF EXITS - The number of unique exit points from this procedure

MISC - The number of executable statements which do not fit one of the above classifications.

COMMON BLOCKS - The number of named COMMON blocks that have been declared for this procedure.

PROGRAM CALL CROSSREF

This screen provides a crossreference of locations that a specified procedure has been called by other modules. An example of this screen is shown in Appendix A-7.

The fields are defined as follows:

PROCEDURE - The name of the procedure which we are crossreferencing

CALLED BY - A procedure which references the subroutine

REFS - The number of times the subroutine is called by that reference.

ASTAR CALL TREE

The CALL TREE screen is used to produce a structure chart for a program or procedure. An example of this screen is shown in Appendix A-8. When asked for a procedure name, enter any VALID procedure name that has been coded. There are several options which you may specify. They are:

;OUT=filename - specify output filename

;LEVEL=nn - specify the number of levels in the tree or structure chart.

;EXT - include references to system intrinsics

ASTAR DATA SET XREF

The DATA SET XREF screen allows the user to determine where a particular dataset has been used within a program or set of programs. An example of this screen may be seen in Appendix A-9.

The fields are defined as follows:

NAME - The dataset name for which we are looking

USED IN - The filename which contains a reference to the dataset

R# - Logical record number of the reference to the dataset

DATE - The date that the file was crossreferenced.

ASTAR $INCLUDE, XEQ, USE FILES

This screen allows the user to locate everywhere a particular file is used as either an USE, XEQ or $INCLUDE file. An example of this screen may be seen in Appendix A-10.

The fields are defined as follows:

INCLUDE - The name of the file being checked

USED BY - A file that uses the specified file

TYPE - Reference to the file was made by either a USE

(EDIT/3000), XEQ or $INCLUDE
R# - Logical record number where the reference was made
DATE - The date that the crossreference was made.

ASTAR UDC COMMAND CROSSREF

This screen allows the user to identify which logon user accesses a particular UDC file, and what commands are available to that user. An example of this screen is shown in Appendix A-11.

The fields are defined as follows:

SOURCE - The name of the UDC file to be examined
COMMAND - A UDC command name
USER - Logon username and account name which can use the command specified in the UDC file.
R# - Logical record number in the UDC file with which the command starts.
DATE - The date that the UDC file was crossreferenced

LOCATING FILES AND PROCEDURES - APRINT

The APRINT subsystem allows the user to locate and print the occurrence of a file, error number, dataset reference, procedure, program, procedure call and more. An example of this screen may be found in Appendix A-12.

The allowable commands are:

ERR= - Locate a specific error number (ERR=2.5 will find all references to PROG-ERR 2.5)
DSET= - Locate a specific dataset (DSET=USER-DETL will find all references to USER-DETL)
FILE= - Locate a file or set of files (FILE=@.SOURCE will print all files in the SOURCE group)
PROG= - Locate everywhere this program is used (PROG=STARTJOB.PROG will find all references to the program STARTJOB.PROG)
PROC= - Locate the procedurename (PROC=GETDATE will locate the subroutine, function or outer block whose name is GETDATE)
JOB= - Find the filename which contains the jobname specified (JOB=CALDUE will find the stream job which will logon with a jobname of CALDUE)
XEQ= - Find all references to the XEQ, USE or $INCLUDE file (XEQ=LOOK will find all references to the QUERY XEQ file LOOK)
CALL= - Find everywhere the specified procedure is CALLed. (CALL=GETDATE will find all references to GETDATE)

APRINT does not upshift the object for which you have requested it to search. However, the information stored in the ASTAR database may be upshifted if the language for the source file is not case-sensitive. This allows for APRINT to be used by FORTRAN, PASCAL and C-type languages.

Also, you must remember that some languages like FORTRAN do not consider blank spaces within a name to be significant, whereas SPL, COBOL and PASCAL do. If the language that the source file normally uses removes extra spaces or characters, so will the ASTAR cross-referencing programs.

You may optionally request the listing be directed to a file by using the ;OUT=filename option.

Problems

Early success in crossreferencing stream jobs provided quite a bit of encouragement, then the realities of life set in. Frequently we would run into programs that appeared to discourage the dovetailling that we desired. For instance, with EDIT/3000, we can specify the file to be editted externally by using the commands:

```
:FILE AFILE=source
:FILE EDTTEXT=*AFILE
:RUN EDITOR.PUB.SYS,BASICENTRY
```

When we exit EDIT/3000, the file is automatically kept back in the original file. On the other hand, the TDP equivalent is:

```
:FILE AFILE=source
:RUN TDP.PUB.SYS;INFO="TEXT *AFILE"
```

With TDP, the file is read in okay, but when you try to exit or keep the file, TDP generates the error message "blank file name". Although this appears to be petty at first, it does become a real problem. One of the tasks we wanted to implement was the creation of master and edit files automatically. This is ONE step to a fully automated environment, held up by a balky program.

Some of the programs we used in the TECH account were modified over time by Kevin Sheely to allow a more versatile use. One program, DOCUMENT.PUB.TECH, had not expected to process more than 32,767 lines of source code. Others, such as REFEREE, were enhanced to allow a longer string to identify the program revision.

PROGINFO, SCANNER and others from the CSL were modified to be more flexible in specifying input and output options.

In SPL and FORTRAN/3000, the procedure relative addresses are displayed to the left of the actual source lines. In COBOL, and now FORTRAN77, this information is displayed in a map at the end. There is no standard which specifies how this information is to be displayed. The presentation of this information is totally in the control of the vendor. As a result, this hampers the development of additional tools, such as symbolic debug packages (yes, HP has TOOLSET, but it does not support all the languages) and data flow/data dictionary rule checkers. Third party vendors who wish to write compilers and support symbolic debug must cope with the lack of information on the format in which variables & symbols are stored within the program file.

Equally as distressing is the state of dictionaries. In general, if you wish to use a vendor's dictionary, you must commit to the use of their 4GL tools. HP allows a Dictionary/3000 user to enter and maintain a dictionary for existing applications, but they do not provide any supported intrinsics to access the dictionary with user-written programs. The lack of intrinsics from HP is cushioned somewhat by the fact they have implemented DICTIONARY/3000 on an IMAGE database. Users of COGNOS' POWERHOUSE dictionary, however, are left out in the cold. The POWERHOUSE dictionary is a binary file. Although it would be possible to decode it, the user would be left with a real maintenance headache. We are almost forced to re-invent the wheel, if we want to integrate a data dictionary with ASTAR.

Closing Remarks.

ASTAR was a monumental task! To date, due to a lack of uniformity in most of the outside vendor tools, the implementation has been restricted primarily to the software tracking aids. For instance, TOOLSET stores its software in a special TSAM format. If you wish to read a TOOLSET-developed program, you must either first convert it to a "flat" MPE file or DECODE the internal TSAM file source storage system. The "INCLUDE/COPYLIB" is not the same between all of the compilers. In order to extract the characteristics of the procedures we are indexing, PROCXREF is in itself a "compiler". So far, we have only implemented the extract code for FORTRAN. About the time we conceived and started to implement ASTAR, ROBOT/3000 was introduced. This appears to the the most complete of the third party packages, nearly meeting all of our original goals.

A good programmer's workbench is desperately needed for the HP3000. As we mentioned earlier in the goals, it is impossible for one person to predict all the different ways the environment would be used. Perhaps we (the HP3000 community) need to form a "Software Tools Interest Group". Such a function could be useful in specifying a standard Tool Interchange Format (TIF). Vendors could then be encouraged to adapt their tools to work with the TIF. The CSL/3000 library could assist by working with the library's authors to adapt their tools to the new format. Whatever the means, let us address this task. We invite your views on this subject.

References

[1] Miller, Edward, Tutorial: Automated Tools For Software Engineering, IEEE Computer Society, 1979, page 2

[2] ibid, page 3

[3] HPToolset Reference Manual, printed 7/82, Preface - page v

Authors

Terrell Haines has been employed as an Electronic Data Processing Analyst for the Boeing Company since 1980. He has been editor, and is currently a columnist, for the Boeing Employees' Computer Society Newsletter. Mr. Haines is also a software reviewer for Design News magazine.

Dennis Heidner received his BSEE degree from Montana State University, Bozeman, Montana. He joined the Boeing Aerospace Company in 1978. Mr. Heidner has written "Transaction Logging and Its Uses", presented at the 1982 HP IUG. He was co-author of two papers, "Transaction Logging Tips" and "IMAGE/3000 Performance Planning and Testing", which were presented at the 1983 HP IUG in Montreal. In 1984, he presented the paper "Disaster Planning and Recovery" at the HP IUG conference in Anaheim, California. Mr. Heidner is co-author of "The IMAGE/3000 Handbook" published by WordWare, Seattle, Washington. He has written technical articles which have been published in several magazines. Mr. Heidner has been a member of the Association for Computing Machinery (ACM) since 1982 and a member of the Institute for Electrical and Electronic Engineers (IEEE) since 1974. He is also a member of the IEEE Computer Society.

APPENDIX

A-1. ASTAR MENU

```
        Rev 1.0-85267  Copyright 1985 By the Boeing Company
        ASTAR - Automated Software Tracking And Reporting

                              ASTAR MENU

01  SOFTWARE MOD. REQUEST          09  ERROR INFO
                                   10  ERROR NUMBER XREF
02  JOB INFO
03  JOB/FILE INFO                  11  QUERY REPORT INFO
                                   12  QUERY REPORT-XREF
04  PROGRAMS
                                   13  INCLUDE/XEQ/USE FILES
05  PROCEDURE INFO                 14  UDC COMMAND XREF
06  PROCEDURE CALL XREF            15  MODIFY MPE(ASTAR) KEYWORDS
07  PROCEDURE CALL TREE
                                   16  ADPAN (SNAPSHOT ANALYZER)
08  DATA SET XREF                  17  APRINT
                                   18  APRGINFO
                                   19  TDP
                                   20  DIRK
```

A-2. ASTAR SR'S

ASTAR SR'S

```
 01 SR #                1       02 NAME          AUDIT LIST
 03 REVISION                    04 TOPIC
 05 USER      H/A'S             06 REQUESTOR      D. TOLER
 07 ASSIGNED TO:TH  08 DATE SUBMITTED 08/27/85  09 SCOPE MINOR
 10 PRIORITY HIGH  11 TEAM CHIEF  DH  12 ASTAR DATE  09/05/85
 13 PURPOSE
    Get a list of equipment not found during an inventory, rather
    than doing a scan by individual Prop#. (exception report)
 14 ACTION
    Use the TRS80 Model 100 to inventory then upload to the 3000
    and run the exception from the 3000
 15 COMMENTS
    Program is partially complete on the TRS 80. A program will
    have to be written for the 3000 to get the exception list.
 16 COMP DATE  01/01/86    17 % COMP.      0    18 CLOSED DATE
 19 MIN HRS 20.0  20 MAX HRS 60.0  21 PROB HRS 35.0  22 ACT. HRS .0
 23 QC DATE           24 ALPHA DATE              25  BETA DATE
 26 RELEASE DATE
```

A-3. ASTAR JOB INFO

JOB INFO

```
01 JOB SOURCE  ANYUSER.REPORTS          02 JOBNAME   ANYUSER
03 ACCOUNT   TEIMS                   04 GROUP    REPORTS
05 FIRST REC.          1             06 LAST RECORD          39
07 REVISION 2.2       08 TYPE   09 INITIALS   10 ASTAR DATE 1/1/85
11 PURPOSE
```

A-4. JOB/FILE INFO

MODE:F ACTION:

JOB/FILE INFO

01 SOURCE QUERY.PUB.SYS

UDC COMMAND	MPE COMMAND	STREAM JOB	R#	DATE
Q	RUN	ANYUSER.REPORTS	26	02/25/86
Q	RUN	AUDITRPT.REPORTS	17	02/25/86
	RUN	CODELIST.REPORTS	17	02/26/86
Q	RUN	CODELIST.REPORTS	82	02/27/85
	RUN	COSTOTAL.REPORTS	13	02/25/86
Q	RUN	FULLMFG.REPORTS	12	02/29/84
Q	RUN	GPTEACQ.REPORTS	15	02/15/87

A-5. PROGRAM INFO

```
                                PROGRAM INFO

01 NAME    ADPAN                  02 SOURCE ADPAN.LIB
03 REVISION 1.1             04 ASTAR DATE  07/28/85
05 PURPOSE
  Analyze snaphsot and program abort files.

06 COMMENTS
  The document file for this program is called ADPAN.DOCUMNT.

MAXDATA   DEFAULT      07 SEGMENTS  13    08 CODE SIZE  142012
09 COMP. JOB
```

A-6. PROCEDURE INFO

```
MODE:F ACTION:
                               PROCEDURE INFO

01 NAME    SUBTRACTDATES          02 SOURCE  ADDATES.SOURCE
03 PURPOSE
  This routine takes two dates in YYMMDD format and gives
  you the difference in number of days

04 INITIALS                       05 TYPE       SUB
06 LANGUAGE        FORT           07 STATUS
08 REVISION                       09 ASTAR DATE  08/26/85
10 FIRST REC.            166      11 LAST RECORD    222
12 LINES OF CODE          23      13 LOOP COUNT       0
14 # OF GOTO'S             2      15 # OF IF'S        3
16 EXPRESSIONS            19      17 # OF I/O'S       0
18 CALLS                   0      19 FORMATS          0
20 # OF EXITS              2      21 MISC             0
22 COMMON BLOCKS           0
```

A-7. PROCEDURE CALL CROSSREF

```
MODE:F ACTION:
                                PROCEDURE CALL CROSSREF

   01 PROCEDURE  GETUSERINPUT

     CALLED BY        # REFS
  GETCALLAB              1
  GETCONTROL             1
  GETDAYOFWEEK           1
  GETFEATURES            1
  GETFIELDNAME           1
  GETFOCAL               1
  GETHEADING             1
  GETJOBNAME             1
  GETMAIL                1
```

A-8. ASTAR CALL TREE

```
ATREE  REV  1.0 -85265   (C) 1985 The Boeing Co.
PROCEDURE NAME?ERRLOG

 ATREE  REV  1.0 -85265   (C) 1985 The Boeing Co.
 LEVEL     PROCEDURE NAME   (Type:#refs)        PURPOSE


  1        ERRLOG           (PROG:  0)

  2                 ARMERRORTRAPS    (SUB :  1)
  2                 COMMITSUICIDE    (SUB :  5)
  2                 RELEASEDATE      (SUB :  1)
  2                 SETUPPROCINFO    (SUB :  1)
                          |              |     |
                          |- called      |     |-- # of reference
                             procedure   |           to procedure
                                         |-- type of procedure
                                             (PROG, SUB, FUNCtion)
MAXIMUM TABLE USED:   30  OF  1000 ENTRIES
PROCEDURE NAME?
```

A-9. DATA SET XREF

MODE:F ACTION:

DATA SET XREF

01 NAME USER-DETL

USED IN	R#	DATE
PDBGENRL.SOURCE	49	08/07/85
CODELIST.REPORTS	45	02/25/86
COSTOTAL.REPORTS	33	02/25/86
USERLIST.REPORTS	39	02/25/86
DELUSER.JOBS	10	02/25/86
COMPARE.BACJOBS	24	02/25/86
BECOIDWA.BACRPTS	40	02/25/86
BECOLIST.BACRPTS	22	02/25/86
BECORPT.BACRPTS	34	02/25/86

A-10. INCLUDE FILE CROSSREF INFO

MODE:F ACTION:

INCLUDE FILE CROSSREF INFO

01 INCLUDE LOOK

USED BY	TYPE	R#	DATE
ANYUSER.REPORTS	XEQ	27	02/25/86
AUDITRPT.REPORTS	XEQ	18	02/25/86
CODELIST.REPORTS	XEQ	83	02/25/86
COSTOTAL.REPORTS	XEQ	14	02/25/86
FULLMFG.REPORTS	XEQ	13	02/25/86
GPTEACQ.REPORTS	XEQ	16	02/25/86
INCALLAB.REPORTS	XEQ	13	02/25/86
MAILABL1.REPORTS	XEQ	20	02/25/86
MAILABLS.REPORTS	XEQ	19	02/25/86
OPTAGS.REPORTS	XEQ	13	02/25/86

A-11. UDC COMMAND CROSSREF

MODE:F ACTION:

UDC COMMAND CROSSREF

01 SOURCE TEM2.UDC

COMMAND	USER		R#	DATE
*LOGON	TEM	TEIMS	1	03/01/86
DBUTIL	TEM	TEIMS	9	03/01/86
E	TEM	TEIMS	12	03/01/86
Q	TEM	TEIMS	15	03/01/86
SORT	TEM	TEIMS	18	03/01/86
HOLDAREA	TEM	TEIMS	21	03/01/86
OFFICE	TEM	TEIMS	24	03/01/86
TECH	TEM	TEIMS	27	03/01/86
LOOK	TEM	TEIMS	30	03/01/86
QA	TEM	TEIMS	33	03/01/86

A-12. APRINT

COMMAND?ERR=2.5

APRINT REV 1.0 -86015 (C) 1985 The Boeing Co.
SOURCE-FILE: READUSER.SOURCE / ERR=2.5

```
C
C     ALL'S WELL THAT ENDS WELL
C
      GOTO 2000
C
C     RECORD THE MISSING MESSAGE TAG IN THE ERRLOG
C
1000  CALL OPENERRLOG(IFILERR,IFOPT)
      WRITE(18,610) USERNO,USERBUF(47)
610   FORMAT(X,"(PROG-ERR 2.5) USER#:",S," HAS A MESSAGE TAG#
     *10X,"THE MESSAGE IS DOES NOT EXIST!")
      CALL FCLOSE(IFILERR,1,0)
C
C      EXIT THIS PROCEDURE
C
2000  RETURN
```

The Bug Stops Here!

Dennis Heidner
Boeing Aerospace Company

I. INTRODUCTION

The cost of software is rising, which is not a profound statement to make when you consider that we have become accustomed to the idea that software (and maintenance) will be 90% or more of the total cost of a computer system. Software is labor intensive, so as the cost of labor rises so does your software cost. But are you getting your money's worth? Software, just like hardware, has a life cycle: first there is the product conception, the investigation of the product and its market, then design, development, product test and finally delivery. But is that it? No! Most studies indicate that the largest cost of the software is AFTER the product is delivered, in what is known as the maintenance phase. (Ever wonder why the monthly maintenance costs for H-P software products are so high?)

Software maintenance generally falls into one of several different categories; they include such areas as adaptive maintenance, perfective maintenance, and simply fixing the outright program bugs. Adaptive maintenance is generally modifications made to the software product so that it remains functional; for instance, the IRS every year spends considerable time adapting their software to match the new tax laws passed by Congress. Perfective maintenance means that the software is being modified to enhance its usability or its position in the marketplace. Both of these types of maintenance generally provide a return on your time investment; however the third category, fixing bugs, simply brings the product up to what it should be, with no additional features. (Have you ever heard of a sales person bragging that they fixed 57 bugs in their product last year?)

Fortunately for most of us, less then 20% of our time is spent fixing program bugs, but would it not be nicer if we spent less

than 5% of our time fixing bugs? [1] In many data-processing shops that translates into one additional head! The purpose of this paper is to present some ideas, which if incorporated into your software, will help reduce the amount of time spent tracking down nasty problems such as program aborts. The paper will cover three areas, spotting the bug, trapping the bug and finally, killing the bug!

Before we continue on, let me emphasize that the techniques I advocate in this paper are not substitutes for structured design, programming, code walk-throughs or testing! For those readers who would like to learn more about structured design, programming or testing, there is a list of references at the end of this paper. [2] [3] [4]

II. SPOTTING THE BUG

The best time to spot bugs in programs is before the product is out to the user (similiar to cleaning house before relatives visit)! This can be accomplished by establishing a rigorous test plan, which the software must pass before it's released. At the HP3000 International Conference in Anaheim, Dan Coates and Michael McCaffrey from H-P talked about the software quality assurance program that H-P has implemented. The quality assurance lab has developed over 800 stream jobs which contain more than 10,000 separate tests! [5]

Test procedures

Locating bugs is, of course, the goal of product test for several reasons; first the cost of fixing a bug once the product has been released is much higher, and second while in product test you are in a more controlled environment where you can generally locate and duplicate a bug more easily. Notice the general tone of this paragraph: we are looking for bugs, not trying to prove the program works. Let me digress another step and talk about the population of bugs. If you have a program that is one thousand lines long,

and you are very optimistic, you might hope that the program is 99% free of bugs. What this means is that someplace in your program there may still be ten lines containing bugs. If you were out to prove the program was correct, the odds are that it will appear to you that it is, even though there are still a few bugs there! It is important to keep in mind Murphy's law of revelation, which is "The hidden flaw never remains hidden."

The test procedure, really, is a program written in the language of your application program. If your program is designed to control WIDGETS and use V/3000, then the native language of your test procedures is WIDGETS with the V/3000 enhancement. Most universities and colleges offer classes in programming in COBOL, PASCAL, FORTRAN, etc., but to my knowledge, there are no classes taught in programming in WIDGETS! This means that when you write your test procedure it will be a learning experience for your staff. Do not expect to have test procedures which cover all the possible cases. If you miss an important test case, this is really a bug in the test plan! It is not uncommon for the first test procedures to have as many or more bugs in them as the programs themselves!

V/3000 users have one additional problem on their hands: how to test the programs and screens in an automated manner. The only commercially available package of which I'm aware is called VTEST; written by Wick Hill Associates, it is marketed by TYMLABS [6].

It doesn't work!

We must recognize that even if we have a good test plan, there will be some bugs that are not caught. This brings up the next way that bugs are discovered: the user calls up and says, "It does not look right!". My initial response to such a general statment is quite negative; however it is our job to turn around the general reports and get the more detailed information we need. This is done by asking more specific questions. For instance, when the user reports that it does not work right, I will normally ask several questions such as: Who are you? What were you doing when it did not work right? What logon name had you used? Has this ever happened before? Is this problem preventing you from working?

Since we do not want to always be grilling our users when they believe they have spotted a bug, we must have a documented procedure for capturing as much information as possible. My first attempt at this was to beg the users to write down the information off the screen, along with the sequence of steps they were going through when the bug occurred. THIS FAILED HORRIBLY! What I found out was that most users have the same aversion for writing that I do, and when they do write, they are prone to transposing numbers.

On many occasions I spent hours trying to locate a bug in the wrong procedure, because the stack marker which was written down was incorrect. The programs at our site are menu-driven, with a feature which allows the experienced user to enter in one step the commands to drop them several menus lower. In other words, if a user wanted selection #1 from the current menu, followed by choice #3 in the next level down, followed by #2 in the one below the second level, the user could enter in: 1,3,2. This is very handy for the users, but a problem for anybody trying to read the scenario that the user wrote down, which looked something like: 1,3,2,4,1,0,3,M,00007635,AC,ME !!!

There must be a better way! The good news is that there are two programs in the contributed library [7], PSCREEN and SCOPY which will copy the information from a screen to a file or the lineprinter. The bad news is that these programs only work with H-P terminals and will operate improperly if the terminal was in block mode. Where possible I set up a logon UDC so when a program aborts, the screen is automatically copied.

Although screen copy routines are a great improvement over relying on handwritten information, they provide only external information to the debugger. When the a bug occurs, what appears on the screen is almost always an imcomplete picture. It would be extremely useful if, in addition to the screen copy, information about the files open, and the values of the program variables could also be saved. After spending a number of hours reading the MPE intrinsic and DEBUG manuals looking for a solution, I found it! The solution is the intrinsic called STACKDUMP. This intrinsic will copy and format the program stack markers and the data area of the stack (anybody who has had a program abort has seen these pesty markers). The person maintaining the program can then use the screen copy, the stack dump, a copy of the program PMAP, a programmer's calculator and a complete listing of the program to locate the bug accurately. Here is an example of a STACKDUMP output:

```
***      STACK  DISPLAY      ***

            S=000070    DL=177644    Z=002266
   Q=000074 P=000010 LCST= 000 STAT=U,1,1,L,0,1,CCG  X=000000

   Q=000062 P=000002 LCST= 001 STAT=U,1,1,L,0,0,CCG  X=000000
   Q=000056 P=000004 LCST= 002 STAT=U,1,1,L,0,0,CCG  X=000000
   Q=000050  P=000033 LCST= 003 STAT=U,1,1,L,0,0,CCG  X=000000

  ..DB..                OCTAL                        ASCII
   00000      000000 000144 000000 177777          .. .d .. ..
   00004      000000 000000 000000 000000          .. .. .. ..
   00010      000000 000000 000000 140032          .. .. .. ..
   00014      000004 000020 040000 000000          .. .. @. ..
   00020      000066 000000 000020 000000          .6 .. .. ..
   00024      000007 172623 031540 000040          .. .. 3' .
   00030      073473 010010 120004 051501          w; .. .. SA
   00034      046520 046105 020123 052101          MP LE  S TA
   00040      041513 042125 046520 020040          CK DU MP
   00044      020040 000000 000034 060304             .. .. '.
   00050      000034 040140 000000 000000          .. @' .. ..
   00054      000005 060303 000006 000000          .. '. .. ..
   00060      000003 060302 000004 177776          .. '. .. ..
   00064      000000 000106 000000 000000          .. .F .. ..
  ** AREA OUT OF BOUNDS **
```

Once the individuals who will maintain the code have taught themselves to how to read program variable maps and program PMAPs, this method of locating bugs is very effective. However it is generally very difficult to teach! This was illustrated to me when I began to explain to another individual in the company how the program collects all this nice information for debugging. The reponse was "How does it work over the phone?" Yes, over the phone! The team that would maintain the software was located some distance from the actual computer hardware. Thus all of our neat stack dumps and screen copies were generally useless!

After a little more careful thought, I realized that generally we do not wish to see the whole stack dump, just selected portions, so why not develope a little program which would read the stack dump from the file, and display only what you asked for? This was the birth of a program called ADPAN [8] (Application DumP ANalyzer).

Due to problems with the STACKDUMP intrinsic, I wrote my own stack dump facility which I call SNAPSHOT. When SNAPSHOT is called it creates a dump file, then copies an exact image of the data stack to the file, along with information on the MPE files which

were open and in use at the time. This snapshot of the process is then later analyzed by running ADPAN.

ADPAN has seven different screens of information which can be displayed; they are: CODE, DUMP, FILES, FILE nn, FLUT, INFO, and TRACE.

The TRACE screen is probably the most important of the screens. This screen displays the procedure names, segment names, p-relative address, Q address and the status for each of the markers in the SNAPSHOT. This allows the user of ADPAN to locate the cause of a program error quickly without needing to refer to a PMAP or have a programmers calculator handy. The TRACE screen looks like:

```
ADPAN 7/83 - Rev 1.1 (C) The Boeing Co, Seattle WA
DUMP: D1921810.PUB.GOODIDEA          PROGRAM: ADEMO.PUB.GOODIDEA

   Q    L   SEGMENT NAME     PROCEDURE NAME   P'REL        STATUS

 00174      ERRORHANDLER     SNAPSHOT         00123 UM,XIN,TRAPS,L,CCG
 00122      ERROR'HANDLER    OVERFLOW         00004 UM,XIN,TRAPS,L,CCG
 00114  ? SL %0173           P'REL = %011026        UM,XIN,TRAPS,L,CCL
 00057      HELP'HELP        OOPS             00005 UM,XIN,TRAPS,L,CCL
 00050      ADPAN'DEMO       PROCEDUREB       00006 UM,XIN,TRAPS,L,CCL
 00044      NEXT'BEST'THING  PROCEDUREA       00002 UM,XIN,TRAPS,L,CCE
 00040      ADPAN'DEMO       SUPERPROGRAM     00035 UM,XIN,TRAPS,L,CCE
 00033  S $MORGUE            TERMINATE'             PM,XIN,L,CCG
```

In this and other examples of screens from ADPAN, the entire line of interest (normally highlighted on HP terminals) is shown underlined.

The CODE screen displays the decompiled code around the PCAL instruction currently being examined by ADPAN. Since not all terminals are capable of scrolling, ADPAN breaks the code down into three regions, and simulates the scrolling programically. Here is a code screen:

```
000004 031003  2.  PCAL 3
000005 004000  ..  DEL ,NOP
000006 031004  2.  PCAL 4
000007 031400  3.  EXIT 0
000010 176031  ..  LRA  P+31  ,I,X (PB+000041)

000011 035002  :.  ADDS 2                  SUPERPROGRAM     <==PROC
000012 004000  ..  DEL ,NOP
000013 021004  ".  LDI  4
000014 033406  7.  LLBL 6
000015 031007  2.  PCAL F'ARITRAP
000016 000707  ..  DZRO,DZRO
000017 021002  ".  LDI  2
000020 172003  ..  LRA  P+3   ,I   (PB+000023)
000021 031011  2.  PCAL FMTINIT'
000022 140005  ..  BR   P+5       (PB+000027)
000023 000014  ..  NOP ,DIVL
000024 044105  HE  LOAD P+105 ,X   (PB+000131)
000025 046114  LL  LOAD P+114 ,I,X (PB+000141)
000026 047400  O.  LOAD Q+ 0  ,I,X
000027 040403  A.  LOAD P+3         (PB+000024)
000030 034403  9.  LDPN 3           (PB+000033)
000031 021005  ".  LDI  5
```

The DUMP screen displays either an area around the current stack marker or a specific region in memory. The user has a choice of OCTAL, HEX, DECIMAL, CHARacter and NOCHARacter formats. The DUMP screen is the default screen. (Any other screen can be requested from the DUMP screen.) For example:

```
ADPAN 7/83 - Rev 1.1 (C) The Boeing Co, Seattle WA, JUL 14 1983
DUMP: D1921810.PUB.GOODIDEA          PROGRAM: ADEMO.PUB.GOODIDEA
Q%000057 P=%000006 X=%000000 STAT=%060703 S=%000071 DL=%177740

ADDR                                DATA
 000036 000047 061305 000005 000000 000003 061304 000004 .'b......
 000045 000000 000007 060705 000004 076400 000000 000004 ....a...}
 000054 000000 000006 060703 000007 000001 010550 111401 ....a....
 000063 000065 000152 111401 000065 140001 000012 135635 .5.j...5.
 000072 000000 001000 000000 000000 000005 177766 000001 .........
 000101 000002 141001 000002 000000 177747 000016 000173 .........
 000110 000052 000004 011027 062573 000035 000001 000115 .*....e{.
 000117 000004 000005 062302 000006 177777 000011 110223 ....d....

>D Q-1;A    'aC'
>D Q-1     %060703
>D Q-1;L     %060703    TRUE
>D Q-1;H     61C3
>D Q-1;I     25027
>D Q-1;D     1640169479
```

Several important items should be noted. The first is that ADPAN will locate and highlight the current stack marker. In our example above this was done by underscoring. Next is that the DUMP screen actually has three separate windows: the header, the data area and the command window. ADPAN uses cursor addressing (if possible) to implement wraparound scrolling within the command window.

The FILES screen allows the user to identify the MPE files that the program had open at the time of the SNAPSHOT. The information displayed includes file number, file name, file options, access options, record size, current record pointer, the number of logical records processed, and the file limit.

```
F#           FILENAME              FOPT%  AOPT%  RECSIZE RECPT
  3 FTN06                          000614 001401    -81     167
  4 FTN05                          000244 001400    -80     167
  5 D1921809.PUB.GOODIDEA          000000 000001    128       3
```

The FILE nn screen allows a user to ZOOM in on a specific file and look at virtually all attributes for the file. In this example we will zoom in on file number five.

```
FILE NAME IS D1921809.PUB.GOODIDEA
FOPTIONS: STD,FEQ,CCTL,F,*FORMAL*,BINARY,NEW
AOPTIONS: WAITIO,BUF,DEF,NOLOCK,SREC,WRITE
RECORD SIZE:    128     BLOCK SIZE:    128    (WORDS)
RECPTR:           3     RECLIMIT:        400
LOGCOUNT:           3   PHYSCOUNT:           1
EOF AT:           3
FILE CODE:      0       # OF USER LABELS:   0
FILE SYSTEM ERROR:      0
```

If the program being examined was written in FORTRAN, the user of ADPAN can request that the FORTRAN LOGICAL UNIT TABLE be displayed; this is the FLUT screen.

```
UNIT F#            FILENAME              FOPT%  AOPT%  RECSIZE RECPT
  6    3 FTN06                           000614 001401    -81     167
  5    4 FTN05                           000244 001400    -80     167
```

The INFO screen lets the user review the general PREP capabilities of the program. In addition the INFO screen displays information on the way the program was segmented, data stack utilization information, and any run-time INFO strings or parms.

```
ADPAN 7/83 - Rev 1.1 (C) The Boeing Co, Seattle WA, JUL 14 1983
DUMP: D1921810.PUB.GOODIDEA          PROGRAM: ADEMO.PUB.GOODIDEA
Q=%000057 P=%000005 X=%000000 STAT=%060703 S=%000071
PROGRAM CAPABILITIES=BA,IA                SNAPSHOT ID: 1

     STACK INFORMATION                  CODE SEGMENT INFO

    DL-DB:     92  7.0%                     5 SEGMENT(S)
    DB-QI:     21  1.6%                  SMALLEST:     8
     QI-Q:     26  2.0%                   LARGEST:   488
      Q-S:     78  5.9%                   AVERAGE:   118
      S-Z:   1096 83.5%                TOTAL WORDS:   592

  MAXDATA:    ??
 MAX Z-DL:  1313

 RUN TIME PARM VALUE:      0
 INFO STRING: ** NO INFO STRING **
```

As you can see, ADPAN provides much more information about the process than the STACKDUMP intrinsic. A common (and very good) practice at a number of HP sites I have visited is to assign an error number to each important step in their programs. Then if there is a problem encountered in that step the program prints out the step number and stops. This is a very simple (but effective) form of defensive programming. Examples of more sophisicated error handling include most of AGADER's functions and the MPE operating system itself. (System failures are MPE's way of preventing further damage by continuing with corrupted system tables.) This process can be enhanced by calling SNAPSHOT, passing it the error number from the program. In this way we can capture the complete environment prior to aborting the program, thus guaranteeing that we always have enough information to properly diagnose the problem.

Databases and bugs

If your application is dependent on a database, then you have a different set of problems. The cause for the wrong information on the screen may be wrong information in the database. One common mistake made by application designers is to assume that once the data has been correctly entered into the database, it will always remain semantically correct. What I mean by semantically correct is that if the weight of a pallet may be between 0 and 30,000 pounds, then a value of -200 is semantically wrong! Another problem can occur when a value from one dataset is used to chain (or point) into another set, but the second entry is missing. Generally when a program runs into such cases (if not anticipated) the results are very unpredictable.

There are three techniques which can be used to locate bugs in our databases before they appear later as bugs in the programs. The first is to write a custom program which checks for and reports semantic errors in the database. For example, database checking programs should verify that items which are defined as dates in the programs contain VALID dates in the database. Fields which contain monetary values or other numeric quantities should be checked to make sure that their range is LEGAL and REASONABLE. Fields which are names of products, companies or individuals should be checked for garbage cnaracters in the fields. Fields which contain phone numbers, addresses or postal mail codes should be verified. Finally if the applications chain from one dataset into another, the test program should do the same. As you might have already guessed, the error check program is a major system in itself. At our site, I run this highly tuned program once a month; its work takes more then six hours!

The second method to locate errors in the database involves active checking for semantic errors by all the application programs. The way this works is that after the user enters in the account number or part number, the program validates all the information related to that number BEFORE the information is displayed. This method assures that before the user is aware that a problem exists, the program has a chance to detect and correct it. This is the method that I use on our main application for the computer.

The final method uses a checksum or hash total for each entry in the database. The application programs, as a next-to-last step before updating the database, generate a checksum for the entity in question. This checksum value becomes an integral part of the item. When the reporting programs read the entry at a later date, they only need to recalculate the checksum value and compare to make sure that they are the same. This technique is most useful for detecting changes made in the database by unauthorized programs or QUERY. Unfortunately if the error was made before the checksum was generated the first time then it will not be detected later. An example of the use of a checksum to detect unauthorized changes is in the file labels on the HP3000.

When I first started writing programs which accessed IMAGE databases, I would generally check the status of the IMAGE intrinsics, then call DBEXPLAIN. After the first time a user wanted to know what all the clutter about dataset so-and-so was, I made an effort to remove the calls and replace them instead with a routine which opens up an error log file, calls DBCALL [9] to get a readable explanation of the problem, then calls DBERROR to obtain the intrinsic name, database name and dataset name. A final call is made to DBSTATUS [10], then all the available information is written to the error log file. For example:

```
==>ZEP  .ZESTY  ,DATA   LDEV:43 #S81  TUE, MAY  1, 1984 8:01P
Rev 2.00-84114 PROGRAM: TESTPROG P=%014.002514  Q=%015263
(PROG-ERR 2.29) Internal application or data base error
DBGET   mode 5  on SPECIFICATION of PAZAZZ opened mode  1
END OF CHAIN
DBSTATUS:  15 .....  %00452  1/ 405 %010076 %015032 5 %004601
 SET: SPECIFICATION:   ITEM-NAME: MODELCODE;
 CHAR. EQUIV OF ITEM: 0003FIDDLE
 DEC.  EQUIV OF ITEM:  12336  12339  17993  17476  19525   8224
```

Remember I said that I generally checked the status of IMAGE calls? Not long after our application was up and running a number of strange errors occured; apparently somebody had used QUERY to delete several entries that the programs always expected to be there. Since the program did not check the status of the previous

IMAGE call, it did not detect the problem. The end result was a bug which migrated throughout the database and took several days to track down! Always check the status to make sure it is acceptable!

Who did it?

If we have detected an error in the database, how do we locate the cause of the problem? Hewlett-Packard has provided database users with the ability to log transactions made to an IMAGE database to either a disc file or a magnetic tape. This record can then be replayed at a later date either to recover after a system failure, or in the case of bugs, to audit the database. There are currently two programs available which can be used to audit the log, DBAUDIT and LOGLIST [11] [12] [13] [14].

III. TRAPPING THE BUG

Some times we do not have sufficient warning to set an error number and abort; for example a BOUNDS VIOLATION will generally abort the program and print out the VERY UNFRIENDLY STACK MARKER in the middle of your V/3000 form. In most cases using a screen copy routine or having the users write the information down is ineffective since the stack marker is spread throughout the form. We really want the computer to transfer to our error routines when an abnormal condition occurs. There is a facility to do this; it is called USER TRAPS.

Choosing the right trap

User traps are probably one of the least understood features of the HP3000 computer and its operating system. This is unfortunate when you consider the power they provide to detect and correct program errors. Traps are provided for the following items: [15]

Type of error encountered	Trap intrinsic
Enable hardware arithmetic traps	(ARITRAP)
Floating point divide by zero	(XARITRAP)
Integer divide by zero	(XARITRAP)
Floating point underflow	(XARITRAP)
Floating point overflow	(XARITRAP)
Integer overflow	(XARITRAP)
Extended precision overflow	(XARITRAP)
Extended precision underflow	(XARITRAP)
Decimal overflow	(XARITRAP)
Invalid ASCII digit	(XARITRAP)
Invalid decimal digit	(XARITRAP)
Invalid source word count	(XARITRAP)
Invalid decimal operand length	(XARITRAP)
Decimal divide by zero	(XARITRAP)
Bad stack marker	(XCODETRAP)
Bounds Violation	(XCODETRAP)
CST Violation	(XCODETRAP)
STT Violation	(XCODETRAP)
Illegal address	(XCODETRAP)
Non-responding module	(XCODETRAP)
Privileged Mode intruction	(XCODETRAP)
Unimplemented instruction	(XCODETRAP)
Compiler library errors (55 total)	(XLIBTRAP)
Invalid substring designator	(XLIBTRAP)
Formatter errors (FORTRAN)	(XLIBTRAP)
MPE intrinsic errors	(XSYSTRAP)

Setting the traps

Except in FORTRAN programs the user traps must be enabled by calling the respective MPE intrinsic. When enabling the trap, the plabel for the desired error-handling routine is checked to make sure that it is valid, according to the following rules:

1. If the call to enable the trap was made from a non-privileged program, group SL or public SL, the trap handling routine must also be non-privileged.

2. If the call to enable the trap was made from a privileged program, group SL or public SL, then the trap handling routine may be privileged or non-privileged, in either the program, group SL or public SL.

3. If the call to enable the trap was made from an MPE SL segment, then the error handling routine must reside in any non-MPE SL segment.

Arithmetic errors

For example, the user may enable a trap routine for arithmetic errors by calling XARITRAP as shown below.

```
                IV   IV      I       I
XARITRAP(mask,plabel,oldmask,oldplabel)
```

mask - Bit mask indicating which types of arithmetic errors are to be trapped (refer to the HP intrinsic manual [16]). mask = 0 disables the traps.

plabel - External type label of the application's trap procedure. plabel = 0 disables the traps.

oldmask - The previous bit mask for the arithmetic traps.

oldplabel- The previous external type label of the application's error procedure (0 if not previously enabled).

Example of an SPL routine to enable all arithmetic traps:

```
PROCEDURE    ARMTRAPS;
BEGIN
  INTRINSIC XARITRAP;
  INTEGER OLDMASK,OLDPLABEL;
  XARITRAP(%37777,@ARITH'ERROR,OLDMASK,OLDPLABEL);
END;
```

EXAMPLE of an SPL routine to handle traps caused by arithmetic errors:

```
PROCEDURE ARITH'ERROR;
BEGIN
  ARRAY BUFF(0:40);
  BYTE ARRAY STRING(*)=BUFF;
  INTRINSIC PRINT,QUIT;
  SNAPSHOT(0);
 MOVE STRING := ("Arithmetic error! SNAPSHOT was taken!");
  PRINT (BUFF,-38,0);
  QUIT(0);
<<  WISHFUL THINKING. WE CAN NEVER RETURN THROUGH THE END! >>
  END;
```

Users of FORTRAN have the ability to enable traps selectively by using the "ON error condition CALL subroutine" statement [17]. unique procedures. The trap mechanism in FORTRAN very flexible; it does not come free, though. In order to separate integer overflows from divide by zero, the FORTRAN run-time library plays a few games. Using the ON statement results in a named COMMON called TRAPCOM' being established on your behalf. When an integer overflow occurs, the computer transfers control not directly to your routine, but to a library routine. This library routine then determines the type of hardware trap that was invoked and accesses TRAPCOM' to obtain the plabel for your routine. Once the library has a valid plabel, it transfers control to your error handling routine by placing the plabel on the top of the stack and performing a PCAL 0.

A user may enable traps for integer overflows and integer divide by zero by using the following FORTRAN stetements:

```
ON INTEGER OVERFLOW CALL OVERFLOW ROUTINE
ON INTEGER DIV 0    CALL DIVIDE0  ROUTINE
```

HP sites that are heavy users of COBOL have a completely different story on their hands. COBOL deliberately calls a routine called C'TRAP to enable SELECTED traps. This was done because when a field is MOVEd in a COBOL program, the COBOL library handles any type conversion that is necessary. The traps that C'TRAP enables are:

```
Integer divide by 0
Integer overflow
Decimal overflow
```

```
Decimal divide by 0
Invalid Decimal digit
Invalid ASCII digit
```

One annoying feature of COBOL programs is that when an invalid ASCII character is detected while moving a character field to a numeric field, the COBOL run-time library attempts to "fixup" the mistake (this was done to be compatible with users who read data generated on punched cards, using overpunching). You may change the traps that are enabled so the program will not attempt a fixup but will instead abort, by using the following SPL routines:[18]

```
$control subprogram
begin
intrinsic quit,xaritrap,print;

procedure snapshot(trapnum);
 integer trapnum;
 option external;

procedure c'trap(trap'type); !This is a variation of the
   value trap'type;          !procedure found in the
   integer trap'type;        !COMMUNICATOR 3000
   begin                     !Version G.01.04 of MPE/V
   integer xreg=x,           !(T-Delta-4 MIT)
           deltap=q-2,       !by Dennis Handly and
           status=q-1,       !John Pavone
           scount=q-5,       ! page 3-11 thru 3-19
           s;
   logical save'op;
   integer array bufw(0:39);
   byte array buf(*)=bufw;
   define cvdb'opcode = ((save'op land %177617) = %20604)#;

   save'op:=xreg;
   if trap'type = %20 then      ! integer overflow
      begin
      move bufw:="segment00x  ";
      return 1;
      end;

   if trap'type = %400 then     ! decimal overflow
      begin
      status.(4:2):=1;    ! set CARRY
      tos:=if cvdb'opcode then
                save'op.(11:2) + %31403          ! get SDEC
           else save'op.(10:2)&LSL(1) + %31401;  ! get SDEC
```

```
   assemble(xeq 0);     ! do stacked exit
   end;

if trap'type = %2 then        ! integer divide by zero
   begin
   status.(4:2):=1;     ! set CARRY
   return 1;
   end;

if trap'type = %20000 then    ! decimal divide by zero
   begin
   status.(4:2):=1;     ! set CARRY
   if scount=1 then return 4 else if < then return 2
                                     else return 6;
   end;

   snapshot(trap'type);
   move bufw := "Internal program error, snapshot was taken!";
   print (bufw,-43,0);
quit(trap'type);

end;
```

```
procedure coboltrap;
   begin
   integer dummy;

   <<aborts on illegal decimal or ascii digit after snapshotting>>

   xaritrap(%37777,@c'trap,dummy,dummy);
   end;
end.
```

Bounds violations

Bounds violations, bad stack markers and invalid instructions may be trapped by the UNDOCUMENTED user-callable procedure XCODETRAP. This routine, which has been around for a number of years, is used by DEBUG and, believe it or not, COBOL! The calling sequence for this intrinsic is:

```
                  I          IV
XCODETRAP(newplabel,oldplabel)
```

newplabel - External type plabel of the application's trap procedure. plabel = 0 will disable the trap.

oldplabel - Previous external type plabel of the application's trap procedure. If the trap was disabled, 0 is returned.

NOTE: XCODETRAP is not in the intrinsic SPLINTR file, therefore do not try to declare it as an intrinsic or your programs will not compile.

FORTRAN users may enable this routine by using the following code:

```
EXTERNAL BOUNDS ROUTINE
CALL XCODETRAP(BOUNDS ROUTINE,IOLDPLABEL)
```

Currently users of other languages such as COBOL must use an SPL routine to enable the trap, such as the following:

```
<< Since we can not declare XCODETRAP as an intrinsic
   we must declare it here so the SPL compiler knows
   that it exists.                                    >>
PROCEDURE XCODETRAP(NEWLABEL,OLDLABEL);
VALUE NEWLABEL;
INTEGER NEWLABEL,OLDLABEL;
OPTION EXTERNAL;

PROCEDURE    ARMTRAP;
BEGIN
  INTEGER OLDMASK,OLDPLABEL;
  XCODETRAP(@BOUNDSVIOLATION,OLDPLABEL);
END;
```

Example of the bounds violation trap routine:

```
PROCEDURE BOUNDSVIOLATION;
BEGIN
  ARRAY BUFF(0:40);
  BYTE ARRAY STRING(*)=BUFF;
  INTRINSIC PRINT,QUIT;
  SNAPSHOT(0);
MOVE STRING := ("BOUNDS VIOLATION!  SNAPSHOT was taken!");
  PRINT (BUFF,-40,0);
  QUIT(0);
<<  WISHFUL THINKING. WE CAN NEVER RETURN THROUGH THE END! >>
  END;
```

Run-time library errors

With the exception of SPL, all of the languages on the HP3000 use run-time libraries. If an error is detected while in the library the user has the option to request transfer to a trap handling routine, rather than to abort the program. The calling sequence for this routine is:

```
                IV        I
XLIBTRAP(newplabel,oldplabel)
```

newplabel - External type plabel of the application's trap procedure. plabel = 0 will disable the trap.

oldplabel - Previous external type plabel that was in effect. If the trap was disabled, 0 is returned.

FORTRAN users may enable this trap by using the statements:

```
ON INTERNAL ERROR CALL LIBRARY ROUTINE
ON FORMAT ERROR   CALL LIBRARY ROUTINE
```

Currently users of other languages such as COBOL must use an SPL routine, such as the following, to enable the trap.

```
PROCEDURE    ARMLIBTRAP;
BEGIN
  INTRINSIC XLIBTRAP;
  INTEGER OLDMASK,OLDPLABEL;
  XLIBTRAP(@LIBRARYROUTINE,OLDPLABEL);
END;
```

An example of a library trap routine:

```
PROCEDURE LIBRARYROUTINE;
BEGIN
  ARRAY BUFF(0:40);
  BYTE ARRAY STRING(*)=BUFF;
  INTRINSIC PRINT,QUIT;
  SNAPSHOT(0);
  MOVE STRING := ("LIBRARY error! SNAPSHOT was taken!");
  PRINT (BUFF,-36,0);
  QUIT(0);
<<WISHFUL THINKING. WE CAN NEVER RETURN THROUGH THE END! >>
  END;
```

MPE intrinsic errors

Almost any abnormal condition which occurs within the MPE intrinsics can be detected by using system traps (XSYSTRAP). The calling sequence for this intrinsic is:

```
                IV        I
XSYSTRAP(newplabel,oldplabel)
```

newplabel - External type plabel of the application's trap procedure. plabel = 0 will disable the trap.

oldplabel - Previous external type plabel that was in effect. If the trap was disabled, 0 is returned.

FORTRAN users may enable this trap by using the statement:

```
ON SYSTEM ERROR CALL SYSTEM ROUTINE
```

Currently users of other languages such as COBOL must use an SPL routine to enable the trap. An example of an SPL enabling routine is:

```
PROCEDURE    ARMSYSTRAP;
BEGIN
  INTRINSIC XSYSTRAP;
  INTEGER OLDMASK,OLDPLABEL;
  XSYSTRAP(@SYSTEMROUTINE,OLDPLABEL);
END;
```

An example of system trap routine:

```
PROCEDURE SYSTEMROUTINE;
BEGIN
  ARRAY BUFF(0:40);
  BYTE ARRAY STRING(*)=BUFF;
  INTRINSIC PRINT,QUIT;
  SNAPSHOT(0);
  MOVE STRING := ("SYSTEM error! SNAPSHOT was taken!");
  PRINT (BUFF,-36,0);
  QUIT(0);
<<  WISHFUL THINKING. WE CAN NEVER RETURN THROUGH THE END! >>
  END;
```

A bug! Catch it!

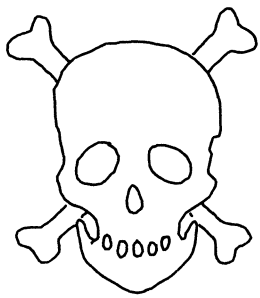
When an error occurs, the hardware transfers control to the correct trap, if it was enabled, otherwise the computer enters standard H-P abort routines. The user-written error handling routine may be in the program, the group SL, or the public SL. User traps

are usable from all languages currently available for the HP3000; however there are some special considerations for COBOL and RPG programs [18].

The error handling routines can be written so that they either attempt to correct the problem (COBOL does this with Invalid ASCII digits) or abort the program. Regardless of which is done, be sure that as much information as possible about the cause of the error is written to a separate error log, so that the bug can be easily corrected.

IV. KILLING THE BUG

Once the process information has been saved or printed, we can abort the program (if desired) in a manner I call STRUCTURED PROGRAM FAILURES. This means that we abort the program in a clearly defined and orderly manner. For instance our abort routine switches the terminal back to character mode, prints a standard abort message on the user's terminal, displays the procedure name in which the bug was detected, then prints an abort message on the operator console (so special program recovery steps can be taken if necessary). A message is sent to any user who is logged on to the programming account, the JCW CIERROR is set to 976 (program abort), JCW is set to FATAL, and finally the program calls QUIT to abort the whole process tree (if any).



Here is an example of a FORTRAN abort procedure, which illustrates the above:

```
$CONTROL MAP,LOCATION,LABEL,STAT
      C
      C  F SUDDEN DEATH:  The purpose of this routine is to provide
      C                   a means of a structured program failure
      C                   similiar to HP's SUDDEN DEATH intrinsic.
      C
      C                   This routine DOES NOT halt the machine or
      C                   cause SF's, it does abort the process
      C                   tree!
      C
      C                   There are two passed variables for this
      C                   routine, IERR and PROCEDURE.
      C                   The  IERR contains the programmer-
      C                   assigned step number, which is included
      C                   in the SNAPSHOT and printed out when the
      C                   program aborts.
      C
      C                   The value of PROCEDURE is a character
      C                   string which is printed on the user's
      C                   screen, and the operator console.
      C                   A corresponding JCW name is checked and
      C                  decremented.  If the resulting JCW is
      C                  greater then zero, this routine will
      C                  return to the calling process.
      C
      C                   In addition, this procedure checks for a
      C                   JCW called DEBUG; if it exists, and > 0,
      C                  the the procedure calls, the H-P
      C                  program debugger.
      C                      written by Dennis Heidner
      C
            SUBROUTINE  F SUDDEN DEATH(IERR,PROCEDURE)
            CHARACTER PROCEDURE*16,COMIMAGE*80,JCWNAME*16
            INTEGER IERR,JCWVALUE
            LOGICAL LTEXT(40),MUST STOP,LJCWVALUE
            EQUIVALENCE (LTEXT(1),COMIMAGE),(JCWVALUE,LJCWVALUE)
            SYSTEM INTRINSIC COMMAND,PRINT,PUTJCW,FINDJCW,DEBUG
            SYSTEM INTRINSIC STACKDUMP,QUITPROG
      C
      C     Take a picture of the data stack....
      C
            CALL SNAPSHOT(IERR)
      C
            DO 100 LENGTH OF STRING=1,16
            IF(PROCEDURE[LENGTH OF STRING:1].EQ.";") GOTO 200
```

```
      IF(PROCEDURE[LENGTH OF STRING:1].EQ." ") GOTO 200
100   CONTINUE
      LENGTH OF STRING = 16
C
C     CHECK THE JCW, WHICH CORRESPONDS TO THE PROCEDURE NAME.
C
200   IF(LENGTH OF STRING .GT. 1) GOTO 300
      PROCEDURE = "NULL"
      LENGTH OF STRING = 5
C
300   JCWNAME = PROCEDURE[1:LENGTH OF STRING - 1]
C
C     DOES THE JCW EXIST?
C
      MUST STOP = .TRUE.
      CALL FINDJCW( JCWNAME, LJCWVALUE, ISTATUS)
      IF(ISTATUS.NE.0) GOTO 500
C
C     DECREMENT THE JCW VALUE
C
      JCW VALUE = JCW VALUE - 1
      CALL PUTJCW ( JCWNAME, LJCWVALUE, ISTATUS)
      IF( JCW VALUE .GT. 0) MUST STOP = .FALSE.
C
C     DISPLAY THE ABORT MESSAGE
C
500   COMIMAGE="Program error in procedure: "
      COMIMAGE[30:LENGTH OF STRING] =
     &  PROCEDURE[1:LENGTH OF STRING]
      CALL PRINT(LTEXT,-50,%0)
C
C     NOTIFY THE SYSTEM OPERATOR....
C
      COMIMAGE="TELLOP  Program aborting in procedure: "
      COMIMAGE[40:LENGTH OF STRING] =
     & PROCEDURE[1:LENGTH OF STRING]
      COMIMAGE[40+LENGTH OF STRING+1:1]=%15C
      CALL COMMAND( COMIMAGE, ICOMERR,IPARM)
C
C     DO WE DROP INTO DEBUG FIRST?
C
      JCWNAME="DEBUG"
      CALL FINDJCW(JCWNAME, LJCWVALUE, ISTATUS)
      IF(( ISTATUS.NE.0) .OR. (JCWVALUE .LE. 0)) GOTO 1000
      CALL DEBUG
C
C     SET THE JCW'S   CIERROR TO 976   AND   JCW  TO FATAL
C
```

```
1000  JCWNAME="CIERROR"
      CALL PUTJCW(JCWNAME,%1720L,ISTATUS)
C
      JCWNAME="JCW"
      CALL PUTJCW(JCWNAME,%100001L,ISTATUS)
C
C     SAY YOUR PRAYERS.....
C
      IF ( MUST STOP )  CALL QUITPROG(IERR)
      RETURN
      END
```

After the bug has been detected or reported, make sure that you use sound software maintenance practices and keep a log of the bugs, the work-arounds, and the fixes. This will enable you to provide better estimates of your future software mainenance costs, estimate number of bugs remaining, provide an indispensible diary for others who might later maintain the software and perhaps most important, provide an experience base so that future software products can be clean and free of similiar bugs.

V. EPITAPH

Although it is impossible to eliminate all bugs from software, it is possible to design the software so that it is easy to maintain and self-diagnosing. This paper has covered several techniques, which if incorporated will help reduce the cost of software maintenance.

VI. REFERENCES

[1] Martin, James and McClure, Carma, "Software Maintenance: The Problem and Its Solutions" (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983). p. 4.

[2] Martin, James and McClure, Carma, "Software Maintenance: The Problem and Its Solutions" (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983).

[3] Glass, Robert L. and Noiseux, Ronald A. "Software Maintenace Guidebook" (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979).

[4] Myers, Glenford J., "Software Reliability: Principles and Practices" (New York, NY: John Wiley & Sons, Inc., 197?)

[5] Coats, Dan and McCaffrey,Michael, "Customer Satisfaction through Quality Software", Anaheim PROCEEDINGS, HPIUG 1984, p. 7.

[6] VTEST available from: TYMLABS
211 East 7th Street
Austin, Texas 78701
(512) 478-0611

[7] Contributed Library Tape, Available from:
HP3000 International Users Group
(INTEREX)
2570 El Camino Real West
4th Floor
Mountain View, CA 94040

[8] ADPAN & SNAPHOT, Anaheim Swap Tape, Available from INTEREX. Update on the C0 CSL tape.

[9] Russell, Marguerite (ed.) "The IMAGE/3000 Handbook", (Se (Seattle, WA: WORDWARE, 1984). p. 283.

[10] ibid, p. 283

[11] Green, Robert M., "Auditing with IMAGE Transaction Logging", San Antonio PROCEEDINGS, HPIUG, 1982

[12] Heidner, Dennis L., "Transaction Logging and Its Uses" San Antonio PROCEEDINGS, HPIUG, 1982

[13] Green, Robert M. and Heidner, Dennis L., "Transaction Logging Tips", Montreal PROCEEDINGS, HPIUG 1983

[14] DBAUDIT, Available from: Robelle Consulting Ltd.
8648 Armstrong Road, RR#6
Langley, B.C. V3A 4P9
Canada (604)-856-3838

LOGLIST, Available from: INTEREX (HPIUG)

[15] Hewlett=Packard, "Intrinsics Reference Manual", Part number 30000-90010

[16] ibid. p. 2-199.

[17] Hewlett-Packard, "FORTRAN Reference Manual", Part number: 30000-90040, p. 4-21 thru p. 4-26.

[18] Hewlett-Packard, "Communicator 3000 Version G.01.04 of MPE/V (T-Delta-4 MIT) pages 3-11 thru 3-19.

LESSONS ON USING HPSQL

Dr. John Hinrichsen
Kirke-Van Orsdel, Inc.
400 Locust
Des Moines, Iowa 50398

Abstract

Relational database systems have been recognized as having great potential benefits, and nearly every database vendor is rushing a relational system to market. H.P.'s relational database, HPSQL, has been available for a year, and the first application systems using this new technology have been completed.

This paper reports on the lessons learned while developing an application using HPSQL. Knowing these lessons will hopefully allow you to avoid some pitfalls in your first use of HPSQL. The most difficult lesson is that HPSQL requires a different mental orientation. Since the data manipulation language SQL is nonprocedural, the programs have a decidedly different and simpler structure. With SQL one specifies what data is needed and not how to obtain it; HPSQL decides the best procedure for accessing the data.

There has been considerable discussion about the performance of relational database systems and whether they are truly suitable for production environments. While there is certainly a cost in resource usage for the many services that HPSQL provides, many complaints of poor performance are actually a result of improper use of the relational technology. Several lessons concern techniques that must be followed in order not to guarantee poor performance.

Introduction

Over the past several years a major shift of opinion has taken place within data processing. The debate has shifted from discussing the relative merits of the different data base architectures (relational, network, and hierarchical) to almost universal recognition that relational data base systems offer irrefutable logical advantages. There is a consensus that the relational characteristics are highly desirable; in the computer magazines nearly all ads for data base management systems (DBMS's) proclaim relational or relational-like functions. There is even general agreement that a relational DBMS should support the Structured Query Language (SQL). The American National Standards Institute

(ANSI) has recently published a standard for SQL which largely confirms the de facto standard set by IBM's DB2. In all, there has been a tremendous shift of opinion in favor of relational DBMS's.

Although there is general agreement on what a relational DBMS should be and on the desirability of such systems, there has been a hesitancy to use relational DBMS's for actual production applications. Questions about the performance of existing implementations are by far the major reason. A secondary consideration is the lack of referential integrity in most of the present relational implementations. Thus the popular attitude is: we want relational but it is not yet technically feasible for high-volume production applications. There is some dividing line which partitions applications into those which can benefit from the advantages of the relational technology and those, which for efficiency considerations, should be implemented in another technology.

Where then does HP's new relational DBMS fit into the scheme? Does HPSQL satisfy the current requirements to be relational, and what are its performance limits? What are the strengths and weaknesses of HPSQL?

Before we begin, I will set forth my personal biases. I believe that relational, SQL-based DBMS's will become as fundamental in data processing in the future as COBOL is today. There is really no other choice. Performance and resource usage is always a concern during a shift to a higher level of programming, but these concerns will lose relevance as more powerful and relatively cheaper computers become available. I feel that eventual use of a relational DBMS is inevitable.

Advantages of Relational Systems

The fundamental objective of relational data base systems was to provide a distinct separation between the logical and physical aspects of data management. In 1970 when relational concepts were being formulated, large application systems had been implemented using the first generation of (non-relational) DBMS's and it was becoming clear that a major weakness of such technologies was an overly rigid data structure. Program maintenance was complex and costly since changes were propagated via the data structure to multiple application programs.

Recently E.F. Codd presented twelve rules that a DBMS must satisfy in order to be "fully relational" (Computerworld October 14-21, 1985). The article was timely since the

definition and properties of relational DBMS's had become obscured, and also vendors had begun to promote heavily the report writers, application generators, etc., associated with their products while deemphasizing data management capabilities. Codd's twelve rules serve to reveal the properties that any DBMS needs for good data management and to indicate the inherent weaknesses of non-relational DBMS's.

HPSQL's Adherence to the Standard

HPSQL conforms quite closely to the standard for SQL. Its major failing is lack of support for subselect statements. Most clauses using subselect statements can be rephrased in an alternate way, but it would be very convenient to be able to extract data and load a table in a single SQL statement:

```
INSERT INTO MIS_EMPLOYEES
   SELECT *
      FROM EMPLOYEES
      WHERE EMPLOYEE_DEPT = 'MIS';
```

I have been informed that a later release of HPSQL may support subselects.

Phone Call Analysis System

Kirke-Van Orsdel, Inc. (KVI) is a third party insurance administrator. Primarily, KVI administers and markets group insurance to members of associations. The risk is carried by one of the full-service insurance companies. Thus, KVI has to satisfy multiple parties: the associations and their members, the carrier insurance companies, and the state regulatory commissions. Because of the many special cases and rapid changes, KVI has a strong interest in the flexibility of relational systems.

The insurance business is also data intensive, and KVI currently has 5-gigabytes of disc storage. There is the on-line activity of answering insureds' questions, issuing new applications, and processing claims, as well as the heavy batch processing of billings and remittances. Much of the customer service is done by phone, and KVI handles approximately 6,000 calls per day. As a prototype application for HPSQL, we decided to develop a Phone-Usage System.

A Telamon PBX Engine was purchased in order to feed call detail data from the KVI ROLM phone system into the HP3000. For each call this data includes the date, time, duration, trunk, extension number, and number dialed. Optionally, an

account number can be assigned to the call. The requirements for the phone-usage system were:

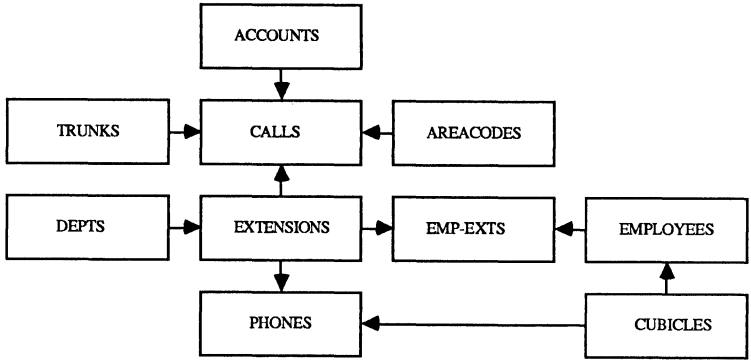
A) Contain the configuration of the PBX system
   1) Trunks
   2) Extensions
   3) Accounts
   4) Equipment inventory and location
   5) Employees and departments
B) Compute management information
   1) Exceptionally long or expensive calls
   2) Summaries by trunks
   3) Summaries by accounts
   4) Summaries by area codes and geographic distribution
   5) Summaries by extension
   6) Summaries by department
C) Satisfy ad hoc requests

Thus the phone-usage system consists of three parts with distinctly different characteristics. Part A) is relatively static with a moderate number of entries, part B) entails statistical information which requires extensive processing to generate, and part C) is satisfying requests normally from detail entries. As previously mentioned, the volume of call detail records is approximately 6,000 per day. The intent of the system was to provide timely on-line information, but there must also be the capability to furnish printed copies of the reports.

We feel that the diverse processing requirements made the phone-usage system a good test for HPSQL. It is an excellent test of the reporting capabilities of HPSQL and of the batch inserting abilities. It is not a demanding test of the on-line updating power of HPSQL.

Designing the System

To start the development of the prototype application, I normalized the data to obtain the following logical data base design:



Each rectangle represents a relational table in the data base, and each arrow represents an external key referencing another table. Phones is a table describing the physical telephone devices, and Cubicles describes locations within the company offices by floor, quadrant of the building, and unique number. All arrows represent one-to-many type relationships. Notice that one employee may have several extension numbers while other employees may share a single extension number.

The maintenance of the tables describing the phone system configuration is done using one form per table. One can page through the table rows in either forward or backward direction and can indicate which rows are to be added, modified, or deleted. The paging is accomplished by means of an SQL SELECT statement for the forward direction and another SELECT statement for the backward direction. These SELECT statements can derive the next or previous screen based on the last or first row on the current screen. These tables contain only a few hundred rows, and I had no difficulty with this portion of the system.

Using SQL, each report is defined by a single SELECT statement. The application program is responsible for deciding which SELECT statement to use and setting the values of any host variable, issuing the SELECT statement, fetching the qualifying rows, and formating the output display. In the present standard for SQL, the qualifying rows can be fetched only once and in ascending order without reissuing the SELECT command. This is adequate for paging forwards through the report but not for paging backwards.

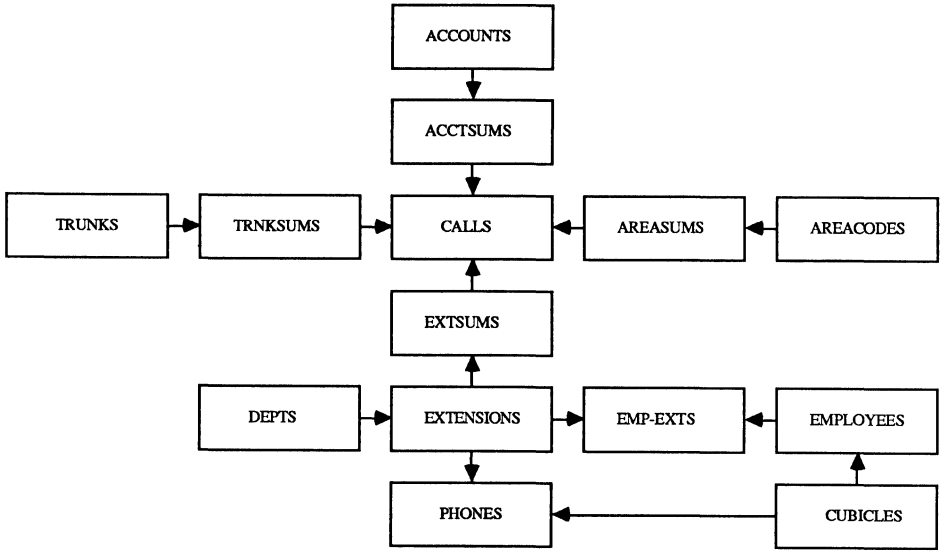
In our application some reports require considerable processing to generate so we do not want to do the processing more than once. Thus, a relative record file was used to store the report while the terminal operator was paging through it. To generate the ten reports a COBOL program was written containing these segments:

1) A list of SELECT statements defining the different reports.
2) For each report, code to fetch the qualifying rows, to format the report lines, and to write them to the relative record file.
3) Generalized procedures for reading records from the relative record file and displaying them as lines in a screen format, and for paging and scrolling forwards and backwards. I tested storing the report lines in working storage instead of a relative record file, but I could detect no material difference in performance. A major reduction of effort is realized by using SQL, since all the file accesses, matching of records, grouping, and sorting is condensed into a single SQL SELECT.

Tuning for Performance

COBOL programs were easily written to load the call records into the data base and generate reports as detailed above. However, response time was several minutes for some of the summary reports. A little thought revealed why this was the case. If you consider the procedure necessary to generate a summary report by trunk for a given day, for example, all 6,000 call entries for that date have to be retrieved, sorted by trunk, summarized by trunk, and finally displayed. The simple SELECT statement masks a great deal of processing. It was decided that 20 seconds would be adequate response time for the reports we were generating. Clearly, if this goal were to be reached it would be necessary to eliminate the on-line sorting of an entire day's call records. It would be necessary to store daily summarizations by account, by trunk, by area code, and by

extension. Additional tables were added to the logical data base design:



Because of the data independence offered by the relational system, these changes were easy to make: I entered the CREATE TABLE statements and altered the SELECT statements in the program to derive the reports from the summary tables instead of the detail calls table.

The load program, which is run nightly to load the day's data into the data base, was enhanced to first insert the detail records into the Calls table and then to execute SELECT statements to summarize the call data and insert the results into the summary tables. Four indexes were placed on the CALLS table based on extension, trunk, area code, and account. After these changes the objective for on-line response time was met, but the load procedures took several hours. In addition, the operations staff complained that the load program used an excessive amount of CPU cycles and impacted other jobs running on the computer. Since KVI was already very short of processing resources, this was unacceptable.

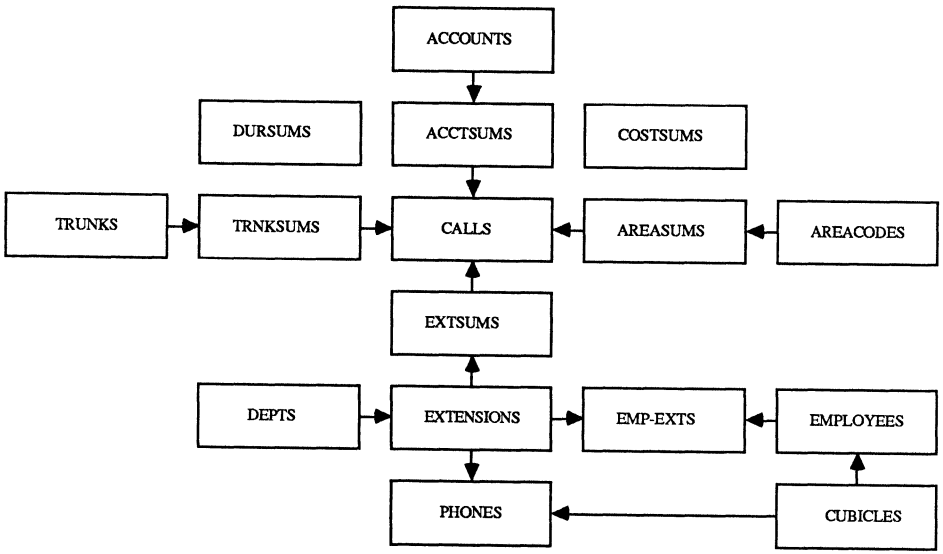
There was no question that the load program had to do a lot of work: insert the 6,000 call rows, maintain 4 indexes, and then extract the 6,000 call rows 4 different times and perform sorts and summarizations. In order to satisfy the

system requirements for the phone-usage system all this work needed to be done, but it needed to be done in a more efficient way. There were three ways in which the load procedures could be shortened:

1) The HPSQL Load utility can load a table in 1/3 the time a COBOL program requires.
2) Sorting and summarizing call information can be done much more efficiently using sequential files rather than extracting all the data from the HPSQL data base.
3) Instead of maintaining indexes on the Calls table to locate the most expensive and longest calls, the top 40 in each category per day could be stored separately.

Seven short COBOL programs were written to implement these loading techniques: one program which extracts trunk and area code information out of the data base into working storage and then computes the cost and state-called for each call record, and six programs which perform internal sorts on the reformatted call records and summarize them by extension, trunk, area code, account, cost, and duration. The output from each of the seven programs is inserted into the data base by the HPSQL Load utility. With these new techniques the load procedure takes about 20 minutes to run.

The final data base design contained two additional tables to contain the longest and most expensive calls for each day:



Querks of HPSQL

With SQL one specifies what data is sought, not how to retrieve it. The HPSQL software is responsible for determining the optimal access route. Until I learned certain tricks, I got unpredictable response times from queries. One request would take seconds while a seemingly equivalent request might take minutes. By examining the selected rows it was evident that in one case an index was being used, while in the other a sweep was being used to retrieve the qualifying rows. A slightly different formulation of the query was causing HPSQL to choose a different access route.

The HPSQL manuals give no guidance on rules one must follow in order to have data accesses use indexes, since such rules are subject to change. The most basic rule is that if you wish to have an SQL command use an index on columns (COL1,COL2,COL3), you should qualify the index columns in their order in the index. For example:

```
SELECT *
  FROM TABLE
  WHERE COL1 = 47
    AND COL2 = 'ABC'
    AND COL3 >= 0;
```

I can give examples in which the order in which the columns appear changes the access route.

Most of my problems where when one of the index columns was defined to be of type DECIMAL(p,s). I found:

a) p must be odd,
b) the COBOL picture must be exactly S9(p-s)V9(s) COMP-3,
c) literal numbers must contain explicit decimal points.

If any of these conditions is violated, the index will never be used, and the request will be processed by doing a sweep.

Properly speaking, the rows of a table are unordered and one cannot predict the order in which they will be retrieved by a SELECT statement unless an ORDER BY clause is present. To correctly use SQL one should use an ORDER BY clause whenever rows are to be presented in a particular sequence. The ORDER BY clause, however, can cause much overhead; all qualifying rows are extracted and then sorted. For example, if you wished to display the call detail records in time sequence, the statement

```
SELECT *
  FROM CALLS
  WHERE CALL_DATE = 870601.
    AND CALL_TIME >= 1000
   ORDER BY CALL_TIME;
```

would cause thousands of rows to be sorted (all entries for calls made after 10 A.M. on the given day). The response would be inadequate for on-line displays. However, if there were an index on the columns (CALL_DATE, CALL_TIME), then the statement

```
SELECT *
  FROM CALLS
  WHERE CALL_DATE = 870601.
    AND CALL_TIME >= 1000;
```

would retrieve the rows using the index, and eliminate the sort. This would be much more efficient if you only wished to see a few entries. (A service request has been submitted to HP to have HPSQL use indexes to satisfy ORDER BY clauses where possible). Although the second approach is more efficient, changing the indexes on the CALL table may alter the order in which the lines are retrieved. In the Phone-Usage System I elected to use the second approach and eliminate sorts in the on-line programs.

HPSQL always reads indexes in ascending order, and has no capability of defining descending indexes. Thus there is no way to retrieve rows in a descending order without performing a sort. For the displays of the most costly and longest calls, which I wanted to appear in descending order, I retrieved the rows from the data base in ascending order and then reversed the order in formatting the terminal screen.

HPSQL inserts new rows at the end of files so as you add and delete rows files tend to fill up. Periodically it is necessary to unload and reload the data base in order to recover unused space. This is done by using the HPSQL Unload and Load utilities to unload each table individually and reload it. It takes approximately 90 minutes wall time to reload 70,000 rows with 10 columns. I use this reorganization procedure to drop out-of-date data by selectively unloading just the current data.

HPSQL Lessons

What have I learned from developing this prototype system using HPSQL? There are several general principles which may help you on your first application.

1) Program coding is definitely simpler and more concise with SQL. You need to code only high-level statements; the HPSQL precompiler converts them to technical COBOL CALL statements.

2) You must check that HPSQL is making optimal use of the available indexes. Slight differences in syntax can cause the indexes not to be used. HPSQL will use the indexes correctly if you observe a few rules.

3) Don't be misled by the conciseness of the SQL statements. Consider the amount of processing that your statements require and be reasonable. Do not expect HPSQL to perform miracles.

4) For large applications it is not sufficient to just normalize the data to design the data base. This is the starting point, but consideration also needs to be given to how the data will be used. In particular, management information systems will generally require summarized data to be stored.

5) HPSQL requires more system resources than does the use of flat files. There is a cost for the benefits of transaction logging, data independence, nonprocedural statements, etc. Perhaps the current release of HPSQL uses 5 to 10 times the CPU cycles compared to a tuned system using flat files. This ratio should decrease with future software releases and with the Spectrum Series hardware.

6) The HPSQL Load utility is several times more efficient than issuing INSERT commands from a COBOL program and should be preferred for data loading.

7) The learning curve for SQL will be longer than you expect. Basically, there are just the four commands; INSERT, DELETE, UPDATE, AND SELECT; but you will be changing from procedural to nonprocedural statements. You will need to rethink your whole approach to program design.

8) For your first HPSQL development it is important to choose an application with moderate amounts of data and reasonable processing requirements. This will allow you to learn to use HPSQL without having to be overly concerned with efficiency and resource usage.

9) HPSQL's potential for improving productivity is so great that I consider its use to be inevitable (or perhaps the use of some other relational data base). But at this time one needs to be selective in the use of HPSQL as it is not yet suitable for all applications. This situation will hopefully change since several other vendors are now endorsing their relational DBMS's for general production use.

10) HPSQL is still an incomplete data base management in that the full complement of integrated application generators, report writers, and utility programs is not yet available. But HPSQL provides a solid base for data management and will serve as the foundation on which to build these future enhancements.

Summary

HPSQL conforms closely to the standard for relational data base management systems. With proper use, HPSQL can provide large gains in productivity on small-to-medium size applications. Caution should be used in attempting to implement large applications using HPSQL, since its performance is not yet as high as that of traditional technologies. You should begin introducing HPSQL into your company as it is the data base technology of the future.

NETWORK MANAGEMENT

Betty Hoo
Hewlett Packard
19420 Homestead Road
Cupertino, CA 95014

What is a Network Management system and why is it important?

To help you understand why Network Management is important, let's imagine the nation's freeways and your city street system as a network. Instead of moving data between two points in a network of computer systems, we are transporting ourselves between a start point and a destination. You can probably picture what would happen if there were no rules of the road, street lights, or signs. This particular network would be in a chaos.

Well, in the same manner that rules of the road and street lights are mechanisms to enforce some degree of control into our freeway and street system, Network Management helps us control a network of computer systems. I will discuss how the need for Network Management evolved and what constitutes a Network Management system as proposed by the International Standards Organization (ISO). With the above scenario in mind, let's look at some of the factors influencing the need for Network Management.

- **AT & T Breakup**

  Since the breakup of the Bell System several years ago, you no longer have one vendor who can be held responsible for the connection from point A to point B. You need a way to manage connections that could involve multiple vendors.

- **Centralized Control of Distributed Systems**

  A historical analysis shows information processing has evolved through three phases. The first phase, centralized computing, is characterized by the concentration of all processing in one machine located at the company Datacenter. The second phase, distributed computing, spread the processing power out to the end users. The proliferation of personal computers over the last several years is an extreme example of distributed computing. The third phase, centralized control of distributed systems, is now in its infancy. Network Management is a tool to realize the benefits of the first two phases--allow MIS to maintain centralized control and still give end users the processing power of distributed systems.

- **Network Growth**

  Networks are growing in size and complexity. As companies grow and the price of computing continues to go down, more and more companies can afford departmental computer systems. Networks are growing in complexity as end users look beyond their own computer system and want to tap the resources available across the network. More often than not, accessing this information can cross multiple vendors' equipment and multiple network link technologies, such as X.25, Ethernet, and 802.3 LANs.

- **Networks as a Company Asset**

  Networks are regarded as company assets because information access is crucial to making timely business decisions. It's not enough now to know that the information exists, but where to find it and how fast can the information be accessed is also critical. There is a direct relationship between information access, timely business decisions, and bottom line profits as information is used as a competitive advantage.

- **Network Management Saves Money**

  Lastly, Network Management saves money. Maximizing network uptime helps protect a company's investment, not only in hardware and software, but also in personnel and the link between sites. There are tangible costs, such as the dollar loss, that can be associated with network down time. More costly can be intangible losses, such as lost business opportunities, due to a lack of or incorrect information to run a business.

These are just some of the reasons why there is a need for Network Management. Let's look at how Network Management will address these needs.

Network Management's objective is to provide medium to large network customers with the tools to create and manage private data networks through all phases of the network life cycle. I will be describing each of the specific "tools" that have been proposed by the International Standards Organization (ISO) as necessary components of a Network Management system. I will also describe the network life cycle and how these tools are used in its various stages.

What makes up a Network Management system? A Network Management system, as proposed by the International Standards Organization, consists of tool sets that can be broken into five categories:

Fault Management: This tool set provides the capabilities to monitor, diagnose, and correct network problems in real-time.

Performance Management: This tool set provides the capability to gather performance statistics to be used for maintaining consistent network performance levels.

Accounting Management: This tool set provides the capability for tracking network usage. These tools are also used for offline problem tracking and maintaining a network inventory.

Configuration Management: This tool set provides the capabilities for centralized management and configuration of remote systems on the network.

Security Management: This tool set provides the capabilities to protect network resources.

Let's look at the tools available in each of these categories.

Fault Management tools help ensure network availability through real-time isolation and resolution of network problems. This tool set can be used to monitor the network and detect, diagnose, and log network problems.

For example, a network error log can be used for troubleshooting. Problem detection tools can include visual or audible operator alarm messages to indicate that an abnormal network condition exists. These messages do not necessarily mean a network component failed but the component is not in its regular operating state. The "failed" component may be a system taken down for preventive maintenance. Path tracing is another diagnostic tool used to isolate network problems. Path tracing can be used to follow the flow of data from its source, through the network, and to its final destination. Through a combination of these tools, network downtime and the disruption to a business' normal operation can be minimized.

Performance Management tools are used to gather statistics on factors that can influence the performance level of a network. These statistics can be gathered over a user-defined time period. Performance Management tools also allow real-time monitoring of performance data. Performance data that can be monitored include response time and resource utilization. Response time is an indication of the time for a command to travel through the network, be processed, and a result returned to the user. Resource utilization statistics can help identify throughput problems or bottlenecks in the network.

Real-time monitoring of network performance data allows operations personnel to take immediate corrective action, such as rerouting data to balance the network load. Performance data collected over time can be used for trend analysis and capacity planning. Performance Management tools help avoid fluctuating network loads to maintain a consistent level of network performance.

Accounting Management tools are used to track usage of network resources. Accounting Management tools can also be used for offline problem tracking and for maintaining an inventory of the network configuration. After defining a cost for the various components of a network, network usage can then be tracked for departmental chargeback.

Departmental chargeback based on connect time is an example of an Accounting Management tool that has been in existence for many years. Service bureaus typically charge customers based on this method. In this case, the charge is assessed for the actual time the customer is connected to a system on the network, whether or not the customer is actively using the system.

Configuration Management tools help provide continuous network operation through centralized control, configuration, and management of remote network resources. These tools are also used to produce tables containing network configuration data, such as device numbers, device types, and physical addresses. The ability to troubleshoot a remote system on the network is one of the Configuration Management tools that can help provide for continuous network operation.

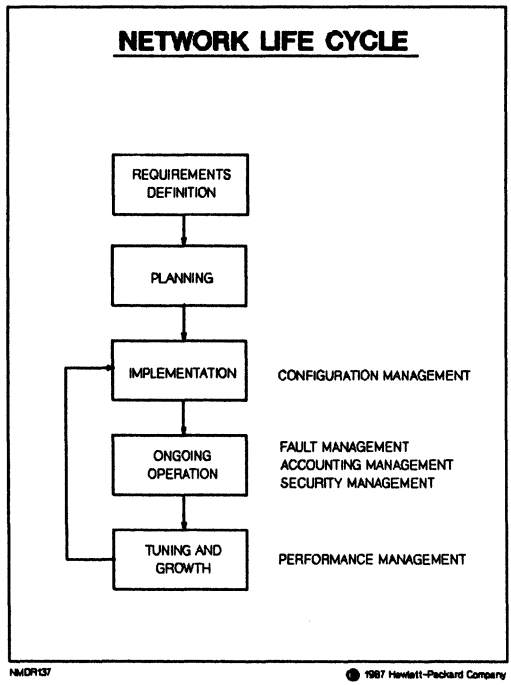
A situation where Configuration Management tools would be useful is a large company with branch locations distributed across the country and a Datacenter located at headquarters. The Datacenter is typically a 24-hour shop with a full technical staff. The technical staff may be able to resolve some remote problems on the network that might otherwise have to wait. The branch office may not be staffed after business hours or may not have the technical expertise. Configuration Management tools minimize the effect a disruption has on the rest of the network.

With the vast amount of data that is stored in a network of computer systems, network level security is needed in addition to system level security to protect all the network's resources. Security Management tools are used to address this need.

One such Security Management tool controls network access. A password security system can be used to restrict unauthorized users from the network. Password security systems usually have several levels. Read, read/write, and execute capabilities are usually associated with the user signon identifier.

Another Security Management tool defines who can control the network from a Network Management standpoint. Staffing to manage a large network of computer systems typically encompasses three job functions. Routine monitoring of the network and basic troubleshooting are performed by a Network Operator. More complex troubleshooting and configuration are performed by a Technical Specialist. Management and planning are responsibilities of a Network Manager or Administrator. Network Management capabilities are associated with these three job functions even though the actual titles will vary by company. For example, it may be appropriate for an operator to be able to run basic network diagnostics but not modify the security structure. Having Network Management capability levels protects the network from the people running it!

Now that you have a basic understanding of the various tool sets that make up a Network Management system, let's take a look at when you would use these tools. The following diagram depicts the various stages of a network life cycle.



Requirements Definition: In this first stage, the customer defines in high-level terms what is to be accomplished with the network.

Planning: This stage involves detailing the requirements and mapping them to the feature set of the networking software. Security and routing structures are also decided.

Implementation: In this stage, the network is installed and tested. Configuration Management tools are used here to configure remote systems and to set up device, address, and routing tables.

Ongoing Operation: Most of the Network Management tool sets are used in this stage. Fault Management tools are used to monitor the network and diagnose problems. Accounting Management tools are used to track network usage for departmental chargeback. Security Management tools are used to protect network resources. Configuration Management tools, though used primarily in the Implementation stage, are also used to a lesser degree in daily operation.

Tuning and Growth: After the network is established, then data collected with Performance Management tools can be used to tune the network. Changes are incorporated by returning to the Implementation stage. Data collected can also be analyzed for trends to predict future growth needs. Continuing this process helps to optimize network performance to reflect changing needs and network loads.

With an understanding of the categories of tool sets that make up a Network Management system, let's revisit the freeway "network" I used at the beginning of my presentation to introduce you to the concept of Network Management. Thinking in terms of the Network Management tools for Fault Management, Performance Management, Accounting Management, Configuration Management, and Security Management, what kind of tools fit into these categories to keep a freeway or city street network running smoothly?

NETWORK MANAGEMENT

EXAMPLE: FREEWAY, CITY STREET SYSTEM

FAULT MANAGEMENT	PERFORMANCE MANAGEMENT	ACCOUNTING MANAGEMENT	CONFIGURATION MANAGEMENT	SECURITY MANAGEMENT
- Traffic Reporters - Highway Patrol - Trouble Telephones	- Traffic Bottlenecks - Speed Limit	- Highway Toll Fees - Bridge Toll	- Traffic Lights - Freeway Signs - City Planning	- Driver's License - License Classes

NMDR139e

© 1986 Hewlett-Packard Company

Fault Management: Traffic reporters and trouble telephones are used for problem detection and reporting. Law enforcement officers monitor the freeways and help restore the flow of traffic after accidents or other disruptions.

Performance Management: A traffic bottleneck can be regarded as a statistic to indicate an unbalanced network load. Analysis showing consistent bottlenecks at one location indicates some corrective measure needs to be taken. Speed limits are also a tool to maintain a consistent level of performance in this network. Though not always obeyed, (as Fault Management tools can detect!) speed limits aid in maintaining a consistent flow across the network.

Accounting Management: Usage of certain resources in this freeway network can be tracked in several ways. Bridges and certain stretches of the freeway network, for example, have toll fees to measure usage.

Configuration Management: Traffic flowing smoothly with the aid of synchronized traffic lights is usually taken for granted until one malfunctions. Traffic lights are an example of remote resources that are managed centrally. The statistics (bottlenecks) collected with the Performance Management tools can be used in conjunction with Configuration Management tools to balance the network load. A City Planning Department, for example, can reconfigure this network by adding an extra lane to alleviate a bottleneck.

Security Management: Lastly, a driver's license is a Security Management tool to protect network resources, such as property and other drivers. A driver's license is similar to a password security system in restricting access to this network to only those users that have passed a test. Classes of driver's licenses further restricts access to the network. From a Network Management standpoint, law enforcement officers responsible for monitoring this network have capability levels or jurisdictions. A highway patrol officer has a different and wider jurisdiction than a city police officer.

Other examples of networks that you're already familiar with are the telephone system and your bank's automated teller machines. What Network Management tools can you think of that manage these networks?

As you may have realized by now, the concept of Network Management and its various categories of tool sets is not new when you look at examples from everyday life. The International Standards Organization is in the process of formalizing the categories of tool sets into a Network Management structure. Having a formal model will aid in defining the future functionality of Network Management tools. Today, the field of Network Management is in its infancy in the computer industry and Network Management still means different things to different people. Many vendors offer Network Management solutions that may, in light of the proposed ISO standards, only address pieces of the Network Management model.

Development of Network Management standards will not only help define future functionality but aid in categorizing existing tools into a formal structure recognized by the computer industry. As the industry moves towards acceptance of Network Management standards, vendors will have the opportunity to integrate their proprietary offerings. Network technology will no longer be a hindrance to information access as Network Management standards will encourage a true multivendor environment. Vendors will have a new benefit to sell, connectivity, but the ultimate winner? You, the user.

ABSTRACT

Building Expert System Shells

Ross Hopmans, Brant Computer Services Ltd.

Now that the MProlog language and development environment has been delivered to the HP 3000, the power of Artificial Intelligence (AI) applications development is availableto Hewlett-Packard's commercial users. This opens the doors to standalone AI applications, knowledge systems interfaced to commercial applications and database, and for the development of task-specific expert system shells.

The focus of this paper is on the development of expert system shells. We examine how artificial intelligence fits into the needs of the business community, look at what expert systems are, and explore the requirements for developing a successful, task-specific expert system shell.

Shells must provide the end-user with more than just the ability to populate the knowledge base with facts and rules. They must provide a strong user interface, a comprehensive explanation facility and versatility within a very specific framework. We examine the meaning of each of these points and look at ways in which they can be implemented.

ABSTRACT

Hybrid Systems - Combining Third, Fourth, Fifth Generation Languages: What Next?

Ross Hopmans, Brant Computer Services Ltd.

Most systems developed today are undertaken using the technology with which the programming team is most familiar. In the short term, this may mean that there is no learning curve to slow down the coding effort; but the reality is that major advances in software technology are being virtually ignored. Every programming language has its strengths and weaknesses. No language is ideal in every situation. Yet a great deal of programming effort goes into forcing languages to be used in applications for which they were never designed. The language for programmers today is to pick the best tool for each individual task - to build hybrid systems to take best advantage of the strengths of third, fourth and fifth generation software technology.

This paper explores the issues involved in building hybrid systems. We examine the types of problems to be solved by business computers such as the HP 3000 and look at how hybrid systems can produce more effective and efficient solutions with less programming effort.

The Power of Graphics in Your Business

Lee Horton
Hewlett-Packard
Personal Software Divison
3410 Central Expressway
Santa Clara, California, 95051

Summary

This paper discusses how graphics help communicate messages by appealing to both verbal and nonverbal thinking. Based on this characterization of thinking, you can use graphics in your presentations and reports to achieve six goals. Each of the goals will be discussed, with graphic examples of how to achieve them.

The paper then discusses the Hewlett-Packard Graphics Gallery, a PC based business graphics package designed to help you acheive the six goals. Charting Gallery, Drawing Gallery, and Executive MemoMaker are described.

Finally, the paper discusses how Graphics Gallery fits into the Personal Productivity Center, including integration between Graphics Gallery and HP 3000 Graphics products. The benefits of graphics on the PC vs the HP 3000 are discussed.

Reaching Verbal and Nonverbal Thinking

The way people think can be characterized in two ways: verbal and nonverbal. The verbal side of thinking uses words to name and define. It counts, keeps track of time, uses reasons and facts to draw conclusions, and has a very step by step approach. Nonverbal thinking manipulates objects, ideas, and concepts, without using words. It puts parts together to make "wholes", perceives patterns, trends and visual images, has no sense of time, and makes decisions based on insight, feelings, and intuition.

When you communicate with others, you will be most effective using the appropriate technique for your message. Messages that require more nonverbal thinking lend themselves to charts, graphs, and pictures. These are effective for showing trends, patterns, and interrelationships. They help the listener to draw conclusions from the whole of your data. If you want your audience to remember specific numbers or draw conclusions by stepping through details, you are appealing to verbal thinking. In this case, use a computer printout; develop a precise table for a report; or for a presentation, create a simple table or text slide that can be used as a visual aid while you explain details.

Six Appropriate Uses for Presentation Graphics

In most cases graphics can help you reach both verbal and nonverbal thinking. Combine spoken or written words with graphics for emphasis, and more detail oriented media for supporting data.

In light of the way people receive your messages, use graphics in your presentations to accomplish these six goals:

- Emphasize and clarify your main point
- Create interest
- Improve recall
- Emphasize relationships and trends
- Save time in analyzing data
- Make translation easier since most pictures are universally understood

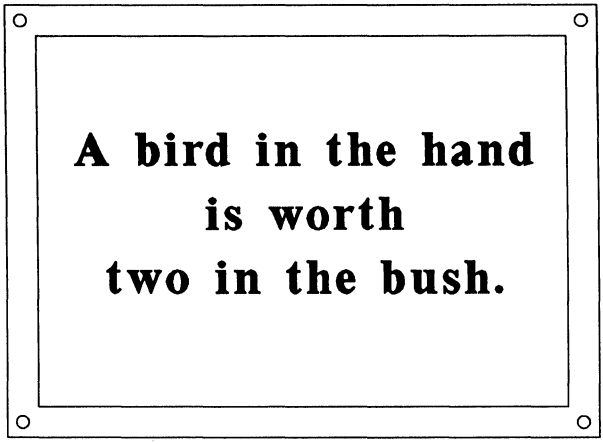
Consider these illustrations of how graphics achieve each of the six goals.

1. Emphasize and clarify your main point

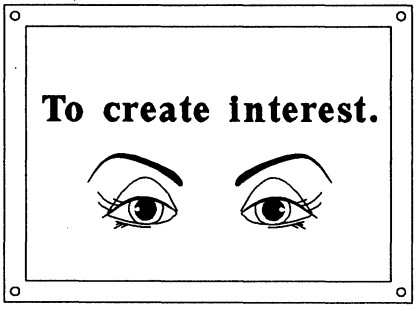
Suppose your main point is, "A bird in the hand is worth two in the bush".

You might emphasize it as shown below:

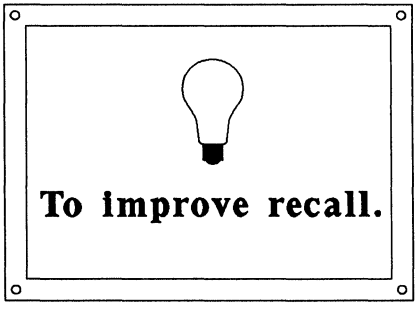
Or simply display your main point in writing as you say it.



This also illustrates two additional uses for graphics:



and

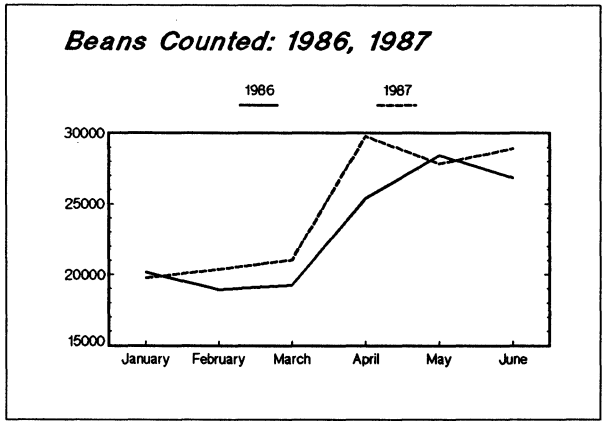


4. and 5. Graphics are useful for emphasizing relationships and trends, and for saving time in analyzing data. For example, suppose your spreadsheet contains the data shown below:

Beans Counted

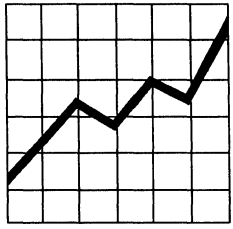
	1986	*1987*
January	20,164	19,750
February	18,937	20,375
March	19,251	21,023
April	25,387	29,758
May	28,438	27,841
June	26,851	28,912

It takes several seconds to see what the chart below shows:



As you can see, there is a two year trend where the bean count increases in the spring.

6. Graphics make translation of ideas easier since most pictures are universally understood. Here is one example.



Meeting the Six Needs: The Graphics Gallery

In support of graphics as an important part of your presentations and documents, Hewlett-Packard has developed "The Graphics Gallery". Graphics Gallery features three products:

Charting Gallery
Drawing Gallery
Executive MemoMaker

This document, including all of the graphics, was created by The Graphics Gallery.

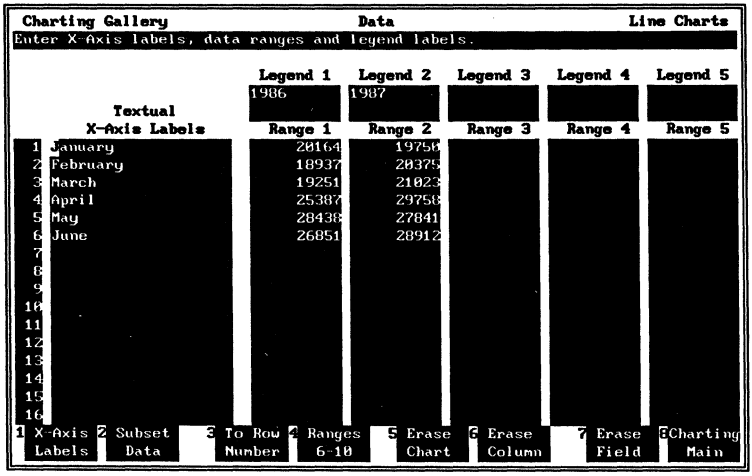
Graphics Gallery provides an easy way to create and edit business graphics that accomplish the six goals. The main philosophy is to provide people in business with professional quality presentation graphics on their PCs.

Charting Gallery

The "Beans Counted" data became a chart using Charting Gallery. Charting Gallery is "data driven" - it creates charts from tables of numbers. You can type the numbers into Charting Gallery, or import them in DIF or ASCII format. This allows you to create charts based on data from almost any applications without retyping it. You can also pull graphs directly from 1-2-3 worksheets, then enhance them in Charting Gallery. Charting Gallery features scattergrams, pie, bar, and line charts. Its charts have been structured by graphic artists using rules for good design, giving good looking results. It has a powerful editor for modifying text fonts, size, and weight, line styles, colors, textures, annotations, etc.

Charting Gallery works as shown below.

1. Type data into the data screen:



You can also load the data in DIF or ASCII formats, or pull a chart from a 1-2-3 or Symphony worksheet.

(Note that the above picture was not created by the Graphics Gallery, and was included in this document by cutting and pasting.)

2. Switch to the "Edit and Draw" screen to view your chart, generated automatically from you data. Use the menu to edit and enhance your chart.

At this point you can plot your chart, or save it in .GAL format to enhance in Drawing Gallery graphics or integrage into an Executive MemoMaker document.

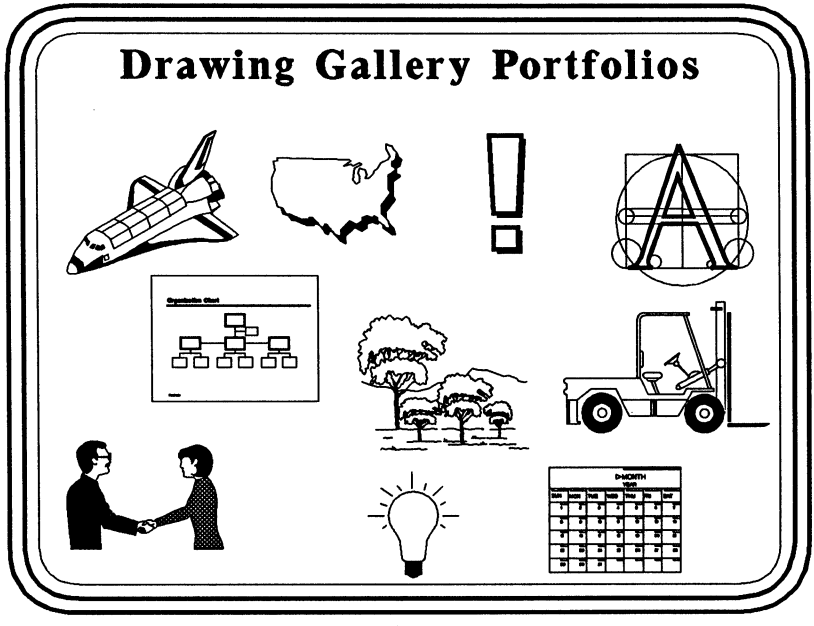
You can also try the same chart data as another chart type, or oriented a new way.

Data driven charts and graphs such as those created by Charting Gallery can help accomplish all of the six goals, but they are particularly useful for communicating what numbers mean, e.g., to emphasize relationships and trends, and to save time in analyzing data.

Remember that the other four goals are emphasis and clarifying, creating interest, improving recall, and facilitating easier translation to other languages. These are where object oriented graphics become useful. Hewlett-Packard has created "Drawing Gallery" to answer the need for such a graphics application on the PC.

Drawing Gallery

Drawing Gallery can load Charting Gallery charts for more free-form editing, or you can create new graphics from scratch. It was designed to create organization charts, text charts, flow diagrams, and illustrations. It comes with a "portfolio" of pictures you can include in your drawings. Here are some examples of the portfolio pictures:

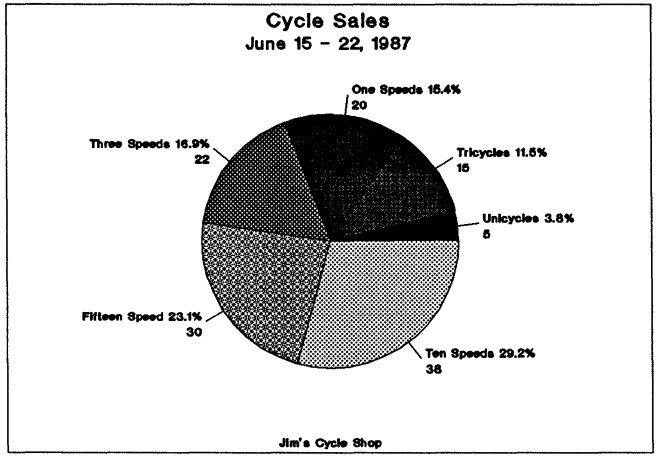


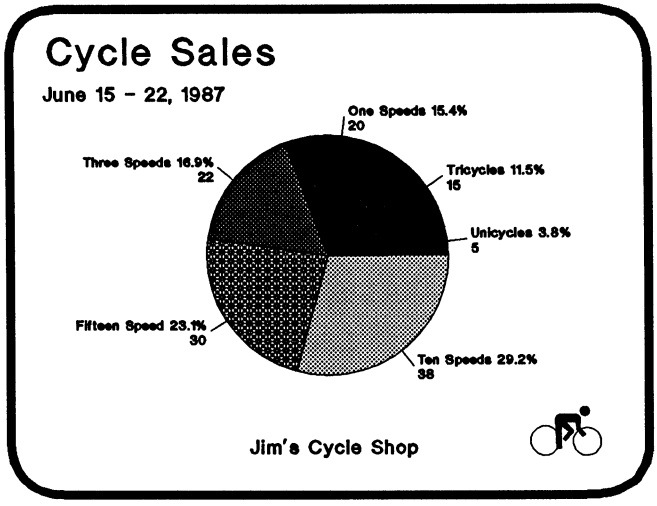
Additional portfolios can be purchased, totalling over 1,800 symbols and illustrations. These include HP Draw Figures, Office Activities, business Management, and Chemical / Petrochemical - thousands of pictures in all.

Drawing Gallery allows more extensive editing than Charting GAllery. It helps you create professional quality pictures by providing a grid as a guide and electronic templates (such as circles, squares, etc.) as building blocks. A wide variety of character and line styles, colors, and sizes are also included. Pull-down menus and your mouse let you produce your drawing quickly and easily.

When you run Drawing Gallery, you first see a menu of options surrounding a screen. The screen contains nothing but a grid. You can choose to load a Charting Gallery chart or an old Drawing Gallery picture, or simply begin drawing. To either edit a chart or old picture, or to create a new picture, add and manipulate graphics by selecting operations from the menus using the mouse or keyboard, then read the screen for further directions. Add pictures from the "electronics templates" by "selecting" them, then "dragging" them into place.

Here is an example of a Charting Gallery chart, on the next page the same chart has been enhanced in Drawing Gallery.





Executive MemoMaker

Finally, you can include your Charting Gallery charts and Drawing Gallery pictures in an Executive MemoMaker document. This paper was created using Executive MemoMaker, Charting, and Drawing Gallery. Executive MemoMaker is a word processor designed to be intuitive and provide the most common word processing functions. These include merged text and Gallery graphics, find and replace, a spell checker, an 85,000 word dictionary, type styles such as bold and underline, and more. To use Executive MemoMaker, just follow the softkey based menu. Most people are productive with Executive MemoMaker within minutes.

So, to review, HP has addressed the six uses for graphics in business by providing Charting Gallery to emphasize relationships and trends and enable faster ways to analyze data. Drawing Gallery helps emphasize and clarify points, create interest, improve audience recall, and help communicate to an audience who speaks varied languages. Finally, Executive MemoMaker allows you to produce reports including graphics.

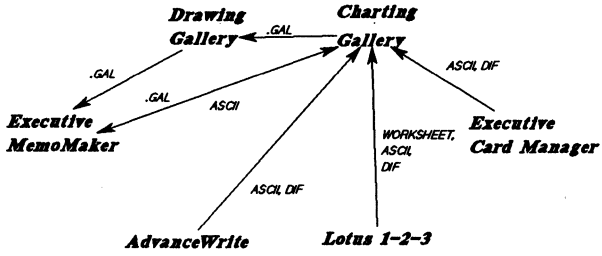
The pages appended to this paper show copies of Graphics Gallery output, plotted to a Hewlett-Packard plotter.

Graphics Gallery and the Personal Productivity Center

Graphics Gallery gives you even more power when you use it with the many other computing tools that comprise your Personal Productivity Center.

On the personal computer level, you can pull data and graphics from the other applications. These include databases, such as Executive Card Manager or R:base 5000, spreadsheets such as 1-2-3 and Symphony, and word processors via ASCII, DIF, and Lotus worksheet file formats. Therefore, you can use your local database to store large amounts of data, manipulate that data using Lotus 1-2-3 or Symphony, then use Gallery to display the results. Finally, put your Gallery pictures in your Executive MemoMaker report. This diagram shows how it all works together.

Vectra Office Integration



All of Hewlett-Packard's strategic PC applications, including Graphics Gallery, are networkable, so you can run them over the HP OfficeShare networks. This way you can share files, printers, plotters, and disc space with other members of you workgroup.

On the HP3000 level, you can use AdvanceLink or AdvanceMail to move your graphics from your PC to the HP3000, then mail them to other people using HP DeskManager.

Hewlett-Packard has also offered a set of business graphics packages on the HP3000, called HP3000 Graphics. These include HPChart, HPDraw, HPMap, and others. Figures created by these applications can be included in documents created by such HP3000 applications as TDP and HP Word, both document processors.

HP Graphics Curator/3000

Graphics Gallery pictures and HP 3000 Graphics pictures can be translated using HP Graphics Curator/3000. HP Graphics Curator/3000 is a graphics converter that runs on the HP 3000. It converts Charting Gallery or Drawing Gallery .GAL files to HP 3000 Graphics FIG file format, and HP 3000 Graphics FIG or DRAW files to GAL format. You can download "curated" HP 3000 Graphics files to your PC, then put them anywhere you would put a Gallery file. You can also upload Gallery files to the HP 3000, "curate" them, and put them anywhere HP3000 graphics files can go.

HP Graphics Curator/3000 also works with HPDeskManager, so that people who receive your GAL pictures can read them while in HPDeskManager. As a result, they will not need to download GAL files to their PCs to view them with Drawing Gallery.

You can run HP Graphics Curator/3000 several ways:

- Interactively, using it's menu driven interface
- From one command line at the MPE prompt
- Programmatically, from MPE UDC's, in job streams, with Advance Mail, and with AdvanceLink command files.

The easiest way is by following the menu-driven interface instructions. Use more automated methods if you have a large number of pictures to convert. Using an AdvanceLink command file, you can also move pictures to and from your PC.

By now you are probably wondering why there are two sets of graphics applications, and why this paper deals mostly with Graphics Gallery.

Why PC Based Graphics?

The HP3000 Graphics applications were first released in 1980. At that time, people did not have easy access to PCs.

Since then PCs have become easier to get and more commonly used. As a result, Hewlett-Packard examined the differences between running business graphics applications on minicomputers and on PCs. They found that PCs provided a significant increase in performance. PC graphics are faster because they can be generated more quickly with the help of the dedicated microprocessor. Because of the large amount of math calculations required for graphics computing, a dedicated CPU significantly improves performance over one which must frequently switch between several

tasks. Since graphics editing requires frequent interaction between the person and the computer, the computer must be able to respond quickly to input. This is also easier with a dedicated PC, because terminals generally work in "block mode", periodically sending blocks of user input to the host computer and receiving blocks of response information. In contrast, PCs accept and respond to one increment of user input at a time.

Graphics performance is also more consistant on the PC than on the minicomputer. The difference is caused by varying number of users sharing a minicomputer at any given time.

Because of performance benefits, plus the fact that PCs became available to people in business, Hewlett-Packard decided to create the Graphics Gallery for the PC. It's PC base allows it to include features that would have been impossible to implement on a multi-user system, especially it's highly interactive user interface. Benchmarks have shown that on-screen performance of Drawing Gallery is at least eight times faster than HP Draw on an empty HP 3000 Series 68. Of course, some of these improvements should be attributed to the fact that Gallery uses new graphics software technology.

Conclusion

Many PC and minicomputer business graphics applications allow you to accomplish the six goals of presentation graphics: emphasis, creating interest, improving recall, showing releationships and trends, data analysis, and enabling translation. However, The Graphics Gallery offers a rich feature set that makes creating professional and effective graphics very easy. It works well with applications in the Hewlett-Packard Personal Productivity Center, and most other popular PC applications. Using the capabilities of the PC, it offers superior performance to minicomputer graphics.

References and Acknowledgements

Dieli, Paula, "HP Combines the Power of The Graphics Gallery and HP 3000 Based Graphics.", **Hewlett-Packard Computer News** (now **Information Systems and Manufacturing News**), March 1, 1986

Matkowski, Betty S., **Steps to Effective Business Graphics**, **Hewlett-Packard Company, San Diego, CA**, 1983

Thanks to Brenda Buchwitz, Claudia Carpenter, Paula Dieli, Joe Malin, and Martha Seaver for their assistance.

Trademarks

1-2-3 (TM), Symphony (TM), and Lotus (TM) are trademarks of Lotus Development Corporation.

R:base 5000 (TM) is a trademark of MicroRim, Inc.

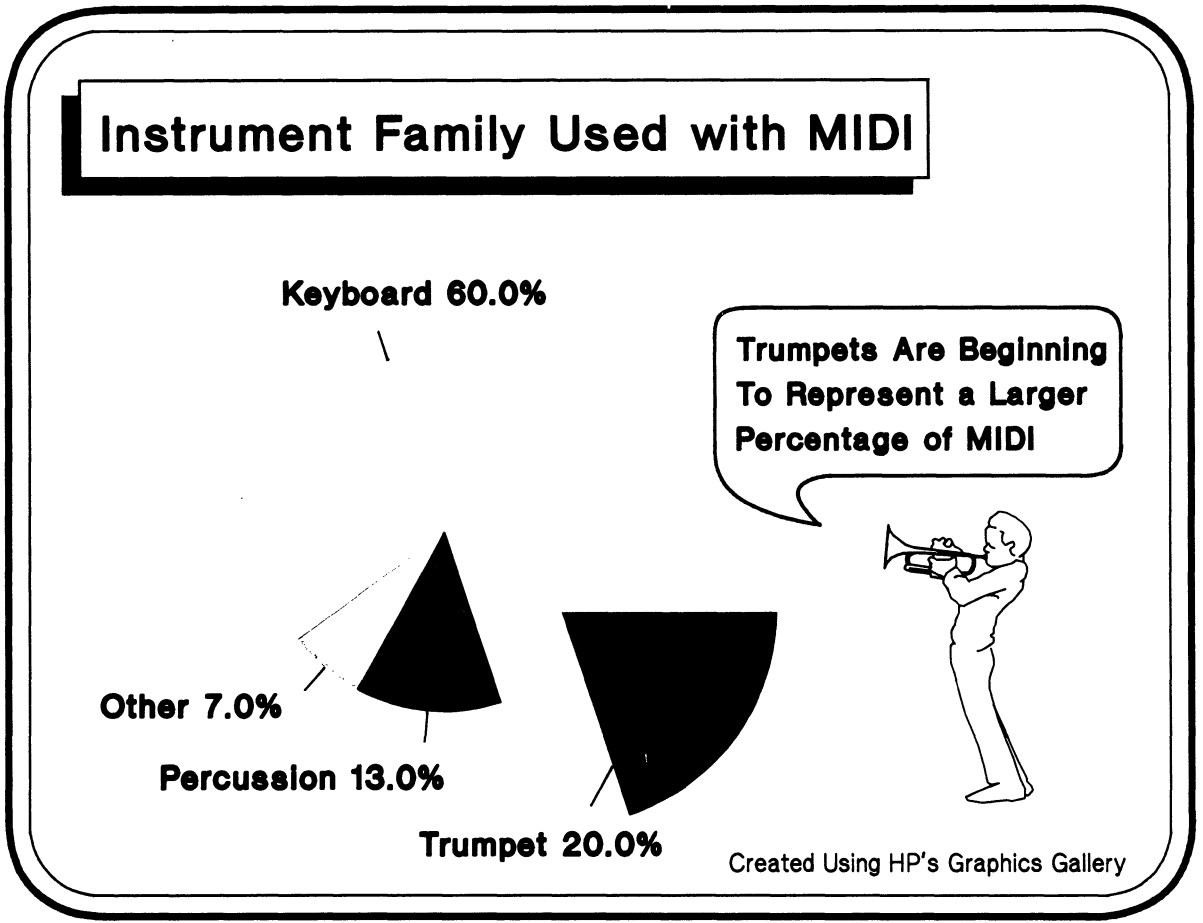
Lee Horton

Lee Horton has worked in the personal computer industry for four years, first in software development, then in technical marketing. She specializes in spreadsheets, graphics, and Hewlett-Packard System Engineer training.

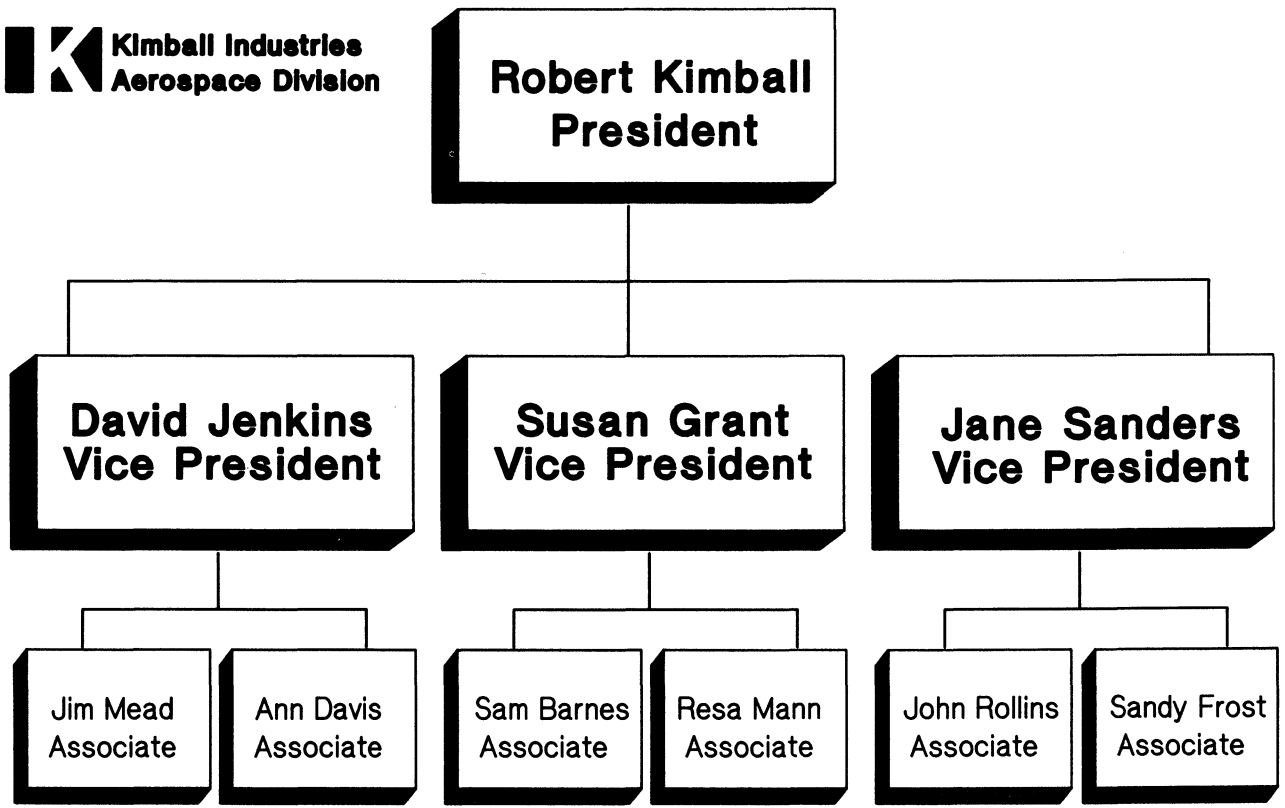
Graphics Gallery Advantages

- Professional quality output
- Complete range of output
  - Full color plots and slides
  - Excellent printed graphics
  - 35 mm output
- Integration
  - Charting and drawing
  - Merged text and graphics
  - Lotus 1-2-3 spreadsheets
- Easy to use

This is an untouched reproduction of a plot created with HP's Drawing Gallery software and an HP 7550A plotter.



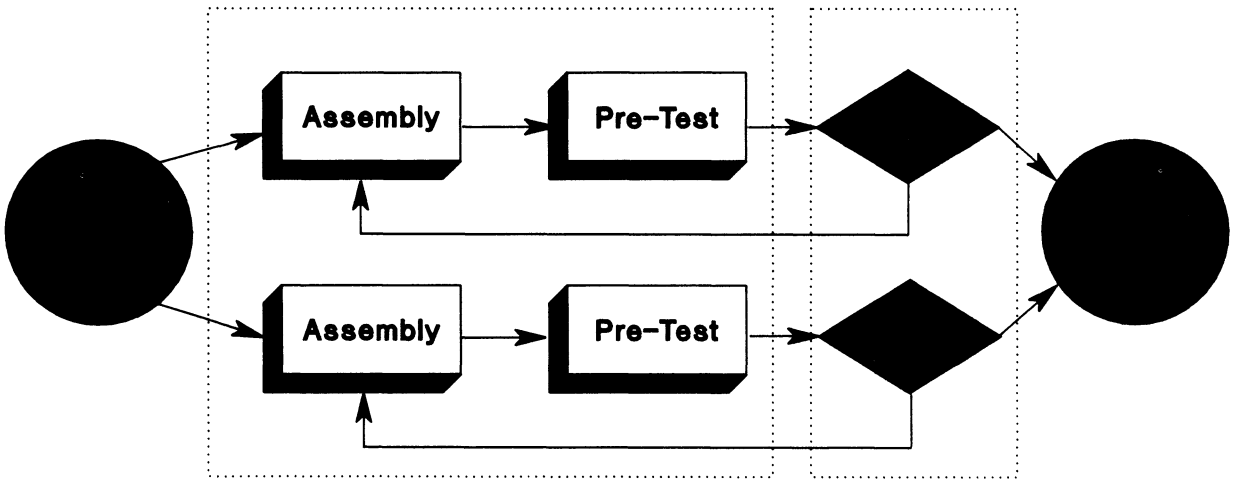
This is an untouched reproduction of a plot created with HP's Charting Gallery and Drawing Gallery software, and an HP 7550A plotter.



This is an untouched reproduction of a plot created with HP's Drawing Gallery software and an HP 7550A plotter.

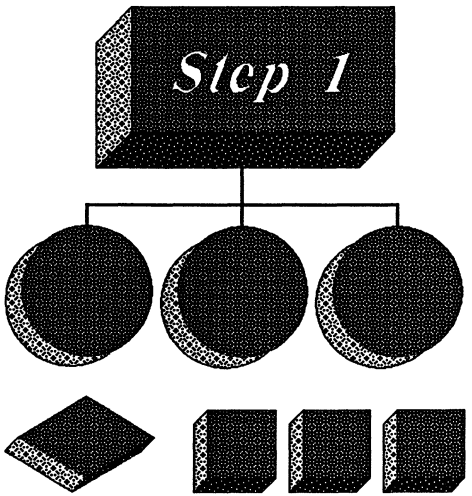
Kimball Industries

New Production Process

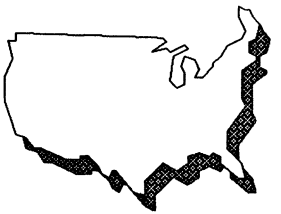


This is an untouched reproduction of a plot created with HP's Drawing Gallery software and an HP 7550A plotter.

Gray Scale by Graphics Gallery



Created Using HP's Drawing Gallery Software and LaserJet Plus Printer at 150 DPI



And Why It's Important:

- **Provides Look of Color on B&W Devices**
- **Objects Can be Overlapped For "Shadow" effect**
- **Maximizes Graphics Capabilities of LaserJet**
- **Ensures Graphic Elements Match Legends**

M E M O

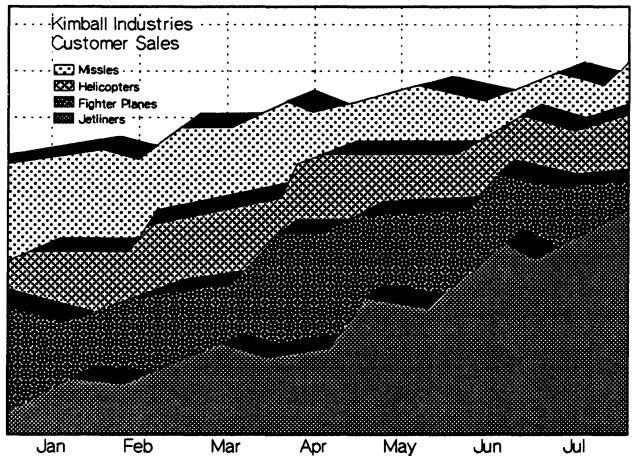
From: Ames Cornish

Date: April 9, 1986

To: Neil Friedman
Dave Obershaw

Subject: Presentation Graphics

As we discussed yesterday, Hewlett-Packard can provide us with everything we need for professional presentation and reporting graphics. To best demonstrate this capability, I have included below the graphic I used last week in our program review.



As you can see, this is the kind of outstanding presentation graphics that we need here at Kimball. There is no doubt that these types of graphics will help us all be much more effective in communicating our thoughts and ideas.

This document was created using Hewlett-Packard's Graphics Gallery and Executive MemoMaker software, and printed on a LaserJet printer.

Sincerely,

Ames Cornish

THE GRAPHICS GALLERY REVIEW

Hewlett-Packard's Graphics Gallery Now Compatible With Industry Leading Desktop Publishing Program

The Graphics Gallery from Hewlett-Packard is a versatile graphics software package that produces truly professional-quality business charts and pictures. It provides excellent results on paper, overhead transparencies and 35mm slides. The Graphics Gallery can also be used for Desktop Publishing.

Professional Results

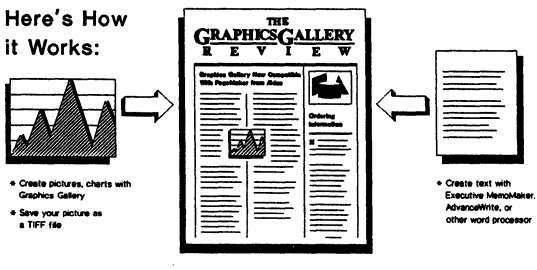
Desktop Publishing lets you quickly create professional-looking proposals, newsletters, brochures and other documents. And for truly powerful and exciting publishing, you need professional-quality pictures like those created by Graphics Gallery. Best of all, you don't need to be an artist to make great looking pictures with Gallery.

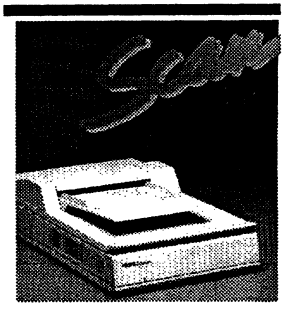
Works With PageMaker

Now the Graphics Gallery works with PageMaker from Aldus. Using PageMaker, Graphics Gallery and a word processor, you can produce professional looking documents. The picture below (created with Graphics Gallery) illustrates how the final document is assembled.

Create text with Executive MemoMaker or AdvanceWrite from HP, or use any of several popular word processors. Create a graph or chart with Graphics Gallery and save it on disc as a TIFF file. Then use PageMaker to place the text and Gallery graphics into your flyer, newsletter or proposal.

Here's How it Works:





Scanning Gallery-- New From HP!!

Now scanned images can be incorporated into your desktop-published documents! With the new HP Scanning Gallery software and ScanJet desktop scanner you can scan images from a broad range of original documents, save the images in industry-standard file formats such as TIFF, and then use the images with PageMaker and other desktop publishing applications.

HP is now offering a comprehensive solution for graphics in desktop publishing: Scanning Gallery and the HP ScanJet desktop scanner for images, Charting Gallery for business charts, and Drawing Gallery for illustrations. For the name of a dealer near you, call 1-800-367-4772.

This flyer was created using Executive MemoMaker, The Graphics Gallery, and PageMaker. The image was scanned using the HP ScanJet desktop scanner. The original was printed on an HP LaserJet using an "F" font cartridge.

ABSTRACT

New Advances In Documentation Retrieval

Doug Iles, Hewlett Packard Company

Where does one get the right information at just the right time? The data processing industry has improved the speed in which end users can access data (e.g. fast report writer, and slick/quick query languages). New demands are emerging. MIS departments are being asked to provide the same access to information that traditionally resides only in large volumes of print, a slip of paper somewhere, or even in someone's head. Computation is not the problem, but searching and retrieval capabilities to find the proverbial "needle in a haystack".

New disciplines are appearing: knowledge engineering, expert systems, on line database services, electronic bulletin boards, electronic mail, etc.

By applying new technilogies, it is possible to make significant improvements in the ability to solve problems and make smarter business decisions. This paper presents solutions available today, together with barriers to implementation.

BEYOND LOGON SECURITY

Joe Junker
Western Savings and Loan
3200 E. Camelback Rd. Ste. 359
Phoenix, AZ 85018

Computer Crime..Computer Security--Buzzwords that have become rampant both in the international press and within our industry. What's the big deal?

How does $500,000.00 sound? That's how much the average computer criminal walks off with, according to William H. Webster, former director of the Federal Bureau of Investigation. The problem may be worse than your board of directors would like to think!

As data processing professionals, we have been charged with the responsibilities for protecting our business from accidental loss and white collar crime. The supportive functions to protect our systems from intentional loss however, are lagging behind. With the increasingly more complex systems, our human ability to manage all of the systems' resources is being taxed to its limit.

Establishing the scope, criteria and initial security plan are the most important building blocks in securing your system from intentional and unintentional loss. Considering the high visibility of system security within the organization, and the potential for loss, beginning a plan for security maintenance and control is an important matter on the data processing manager's agenda.

This paper will present an overview of the HP3000 and security. Summarizing the characteristic operating environment of the HP3000, and providing a view of current access control methods, this paper will discuss application level security in detail, to aid in beginning to develop a security plan suitable for the HP3000.

The Completely Secure System

The principle of the completely secure system is made up of a triad of factors. Physical security is the security derived from computer room access controls, environmental maintenance controls, and communications equipment controls. Operational security is the set of rules and procedures established for operators, tape backups, programming standards, accounting structures, file security matrices, and device restrictions. Application security consists of end-user interfaces and data access capabilities. Theoretically all three of these security types combined and enforced will result in a totally secure system.

For all practical purposes however, no system is impregnable to unauthorized access. Once the first cable is strung outside the computer room, or the first dial-up modem is installed, the data-laden computer becomes a security risk.

Physical, operational, and application security combined can improve the security profile of a system. All three of these components of complete security are made up of five functions: Risk Avoidance, Deterrence, Prevention, Detection, and Recovery.

Security Components

Risk--
R = L * P

The risk equation, R = L * P, is commonly used by risk management consultants in helping clients evaluate the potential for risk (R) based on the expected loss (L) and the probability or frequency of exposure to loss (P). According to the risk equation, the more you have to lose, and/or the higher the probability (or frequency) your system is exposed to potential loss, the higher your total level of risk. Only by reducing the expected loss (L), or the frequency of exposure to loss (P), can risk be minimized.

The HP3000 poses some special challenges to the system manager trying to minimize risk. With its multiprogrammed interactive operating system, and high number of connected or dial-up ports, the frequency for loss (P), can be considered very high.

Operational security set up through MPE file matrices and accounting structures (Engberg,Volokh) can help minimize the expected loss (L) by restricting read or write access from unauthorized persons. IMAGE and MPE security can help "hide" certain elements of data from users based on logon capabilities or location within the accounting structure. Without an additional barrier besides logon capabilities security between the criminal and the data, all the criminal needs is the correct logon to begin destroying or stealing data.

A recent phenomenon has been the introduction of more distributed systems. The divisional or departmental HP3000 lowers the potential loss (L), yet increases the probability for risk (P) by exposing more ports over a wider geographic area.

With distributed systems also come distributed system managers. Some companies (including HP), have turned over the entire accounting structure maintenance within an account to "subsystem managers." With all of this growth in responsibility comes an explosion of system management knowledge which was once locked in the corner of the computer room.

It becomes evident that those of us who use the HP3000 to its best potential, as a distributed processor with 30 to 200 users able to access the system (sometimes 17-20 hours per day) must consider the high potential for risk. We must compensate for this high risk factor through emphasizing the other security functions, deterrence, prevention, detection and recovery.

Deterrence, Prevention, and Detection

When these three functions are discussed, the order in which they actually work together to improve security usually becomes muddled.

Deterrence can start from the moment the user sits at the terminal. Whether logging on, or just hitting return the first thing in the morning, the system should welcome the user with a statement of ownership and intended use. Blake suggests the following :WELCOME message:

```
************************************************************
*                    Welcome to MY SYSTEM                  *
************************************************************
*  This is a private system operated for XYZ Company       *
*  Business ONLY!  Authorization from XYZ management is    *
*  required to use this system.  Use by unauthorized       *
*  persons is prohibited and may result in prosecution.    *
************************************************************
```

Deterrence is meant to combat intent before it has developed sufficiently to become action. Even the smallest deterrence will sometimes keep an honest person honest. The HP3000's :WELCOME facility can be an aid in adding some deterrence to the front end of the system.

Prevention in the HP3000 environment can be found most commonly as the password(s) required to access the system. Lockwords on files, passwords to databases, and user-id's within application programs are other forms of prevention. Where file security is concerned, Volokh contends that IMAGE security and lockword security are not particularly useful. Instead, he suggests that security matrices for files, or application controls in programs control access to databases/files based on the application user's id.

Most preventative measures will be ineffective unless detection is incorporated in the security system. The detection method adopted should not inundate the system manager with information (such as reading all job/session initiation/termination entries in a log file), but should provide enough information to discriminate and deduce intent from the report output.

The auditing function, usually an effective means of corporate fiscal security detection, plays a critical role in the detection of computer crimes. Just as we, the system managers, are having trouble keeping up with the challenges of security, auditing is also lagging behind in the onslaught of technology. In a report prepared by the Stanford Research Institute for the Institute of Internal Auditors, the auditing function was documented as having learned to audit batch operated computer systems, but are not yet able to contend with online, distributed systems with telecommunications access. This can present a real problem in making the detection of computer crime adequate for the HP3000 environment.

As noted above, detection relies on some sort of logging. Effective logging and detection can act as a very strong deterrent. This is where the grand circle of security measures begins.

Deterrence, prevention, and detection interplay with the other functions to begin providing a more cohesive security environment for the system. The best method of securing HP3000 applications and files will combine deterrence, prevention, and detection, and ease their administration.

Recovery

Accurate and timely detection can ease recovery. Disaster recovery, another of our industry's most recent buzzwords, is part of this plan, but not discussed here, since it is usually considered more operational- than application-related. After the accidental or intentional destruction of data, a minor disaster has occurred. Being able to trace either by user, or some other unique identifier the transactions which took place is critical in recovering from such a disaster. It seems again that detection of data modification becomes of paramount importance.

Recommendation

Based on the needs defined thus far for securing the HP3000, it appears desirable to limit the number of people who have direct access to MPE (the colon). By limiting the number of people logging directly into MPE, the following security functions will be affected:

1) Risk is decreased.
   Less people logging on to MPE enables a greater degree of security by posing another barrier for both application and file access to unauthorized persons (there are LESS wide opened doors).

2) Deterrence is enhanced.
   The knowledge that a "big brother" program is managing the users' selection, running, and exits from applications adds an uncertainty to the potential criminal's minds. They will feel that an unanticipated detection is more likely, within such a structure.

3) Prevention is increased.
   Critical applications can be separately passworded and controlled. Even critical transactions, if managed through the security system can be passworded.

4) Detection becomes less cumbersome.
   Logging by application start and stop, or by transaction can be performed. Creating the direct user-to-application cross-reference can aid in the tracking and auditing of users accessing applications.

5) Recovery is made easier.
   With lists created in #4, transactions and file access can be more dependably backed out, or databases restored.

6) It is generally more "friendly" than a colon. Users will have less trouble and make fewer mistakes if the drudgery of typing "RUN PAYROLL" or "PURGE PAYMAST" are taken out of their hands. A mistyped filename in this case constitutes a serious security violation. Would MPE logging tell you this? **NO!**

Understanding the operating environment of the HP3000 has enabled us to narrow our examination of options to only those applications which control user access with MPE behind the scenes. Volokh labels this approach the inclusive approach, where one is permitted only certain specific things.

HP3000 Application Security Administration

Currently, user capabilities without MPE are being controlled by one or more of the following methods:

1) MPE logon UDC's
2) Process-handling (PH) menus
3) Startsess and monitor security systems.

Criteria

We have seen how deterrence, prevention, and detection play important roles in securing a system from loss. Our examination of the three control methods for HP3000 security will be based on each solution's ability to provide those security functions. In addition to the primary security functions, the implementation, integration with existing applications, and administration features of each security method will be reviewed.

The integration, or means of linking the logon security system to the capabilities assigned by applications will be given special emphasis. Beyond logon security, the next most important control points are those included in applications. These security control points can aid in the prevention, detection and recovery from the following security exposures:

1) Accidental Modification

   - Hardware Malfunction
   - Application Software Malfunction
   - Duplication

2) Accidental Destruction

   - Writing over a "good" file
   - Losing an error or success message

3) Intentional Disclosure

   - Reports run under unauthorized circumstances

4) Intentional Modification

   - Adding "unapproved" transactions
   - Modifying vital records (control records)

5) Intentional Destruction

   - Deleting vital records (control records)

MPE Logon UDC's

The MPE logon user defined Command (UDC) in one form or another was the basis for most and is still the basis for most logon security schemes installed on HP3000 hardware.

Implementation

The logon UDC is implemented to initiate an application when the user is logged on, and immediately end the session when the application is ended. Figure 1 shows the "JCL" for a typical logon UDC.

Figure 1

```
MPELOGON
OPTION LOGON,NOBREAK
CONTINUE
RUN CHECKS.PUB.APSYS
BYE
```

The LOGON,NOBREAK in the second line of this UDC disables the break key and immediately at logon instructs MPE to use this UDC for "batch command" input from the rest of the UDC file. By disabling the break key, the user is unable to BREAK and :ABORT the program, which would allow access to MPE. The CONTINUE statement in the third line of this UDC will allow the UDC to complete (logoff the user) should the program abort. In this instance, the CHECKS.PUB.APSYS program is the only program run by the user whose catalog had been set to this UDC file.

The logon UDC is created in an EDITOR file and named by the creator following file naming conventions for the HP3000. The :SETCATALOG *filename* command is used, and MPE cross-references this UDC in CATALOG.PUB.SYS. The :SETCATALOG command has an ;ACCOUNT and ;SYSTEM parameter to allow designation for whole accounts or the entire system for enabling UDC's at logon.

Deterrence

The logon UDC offers little, if any deterrence to criminals. Unless prefaced with the :WELCOME message discussed earlier in this paper the logon UDC is a welcome sight to criminals, putting them directly into one of your organization's applications.

Prevention

The passwords (user,group,account) if implemented are the only barriers to the criminal accessing one of the system's applications. In some cases, implementing UDC's may be less secure than leaving MPE available to users because of the ease of starting an application with logon UDC's. To protect from insider theft or destruction, however, the logon UDC is a good means of preventing either unintentional destruction of files via :PURGE commands, or intentional breaches of security within the operational limits of the MPE accounting structure.

Detection

Little means of detection is available when using application-only logon UDC's. The only available log of system use are session initiations and terminations buried in the voluminous system log files. With such inadequate detection capabilities, deterrence becomes even weaker. Users, and eventually the criminal learns that little control is enforced on their sessions.

Integration

With the single application UDC, there is no integration of multiple systems within the umbrella of a menu or supervisor. It amounts to one logon/one application. For a user to run 3 applications controlled under single application UDC's 3 logons would be needed.

Administration

Since uniform naming of UDC files and keeping track of the relationships of user to UDC would be a manual task, maintenance of the cross-references are not maintained as they should be. Auditors would not be impressed by the effort needed to cross-reference user to UDC's to applications, a task basic to the proper administration of security systems.

Summary

The single application UDC is the weakest form of implementing the inclusive approach to applications-security. Its use would be suitable for HP3000's with only one application allowed per user, but becomes impractical when more than one application is available to a single user.

Process Handling Menus

The glaring weakness of inflexibility in single application logon UDC's were addressed with the development of Process Handling (PH) menu systems. The PH supervisor program, driven by pre-established parameters based on the user's logon or ID, presents a menu of applications available to the user. This approach began to bring HP3000 application security of age. By requiring parameters to run the PH menu program, the implementation of this menu system improved the auditability of user/application relationships on a system-wide basis.

Implementation

The PH supervisor program is inserted into the MPE logon UDC as its single application. This program, capable of process-handling to any number of applications was no longer a limitation for applications security administrators, but a gateway to any of the applications pre-configured for the user. A database is constructed by the security administrator and simple menu screens are created either by the user, or dynamically within the menu program. Cross-references are established in the security database between users and screens. Additional passwords, encrypted, can be added for users to reply to at logon.

Deterrence

There is more visible control over the application environment using PH menus. Deterrence is improved by virtue of this appearance. Another greeting message perhaps customized by application group can be build into the screens displayed to the user.

Prevention

Most PH menu systems commercially available have added passwords to enhance the prevention aspects of security. Most also enforce port security if so desired a well-recognized weakness in MPE security. Enhancements to identify users more specifically, by session name, date, or time of day are part of most PH menu systems. Aging passwords, and notifying users of required password changes are available with some systems.

Detection

With the supervisor program monitoring access into the HP3000, PH systems are capable of a greater degree of detection. Notifications such as printed reports of violations routed to the system manager enhance the system manager's ability to find out about security violations immediately. Detection and follow-up on even the most minor password errors can show users the system manager's commitment to a secure environment.

Integration

Certainly, the PH system is an improvement over UDC's where integration is concerned. Complete integration with the protection of applications on an application level has not been addressed by the PH menu system. Controlling access by "application group" is the strength of PH menu systems. When additional application-level restrictions need to be applied, the individual application subsystem must track and control those.

Administration

The necessity for creating the application environment for each user before implementing PH menu security has improved the accessibility of user profiles for HP3000 logon. System managers must do more planning before implementing PH menu systems, and this has been beneficial to the documentation process. Administering users in MPE, the PH subsystem, and then applications however, complicates security administration. Sometimes 3 or 4 lists of users, user-id's, and application security schemes must be combined to produce one document for auditing the current user access situation.

Summary

PH menus, combined with the reporting of the PH menu parameters, have enhanced application level security tremendously. Although complete integration between MPE, the menu system, and application restrictions has not yet come to the forefront, enough integration to aid security administrators in the documentation of "who can do what when" has been built into PH menu systems. Complete integration of MPE user assignment, PH user assignment, and application user capabilities would be the next logical step in improving the control and auditability of the HP3000 applications environment. Only by bringing the security administrator's job (including MPE user creation and maintenance) under the umbrella of the PH menu system maintenance, can the administration of PH menu systems be improved. Some PH systems have already integrated the :NEWUSER and :ALTUSER commands into the PH User Maintenance. In some instances, the prevention afforded by PH menu systems is more effective than MPE security due to the encryption and random selection of passwords.

STARTSESS and Monitor Security Systems

STARTSESS and monitor security systems rely on an "initiator" program to initiate the user's interaction with applications. With either of these systems, a user's interface with MPE can be limited very effectively. Even the :HELLO command can be eliminated.

Implementation

Either through a batch monitor, or an online monitor program devices are allocated by these security systems. The programs either initiate sessions (STARTSESS) or direct I/O (monitors) to devices configured into the monitor control parameters. The emphasis of security administration using these methods is based on devices rather than users or user-id's.

Deterrence

From the user's perspective, the STARTSESS and monitor systems may appear very similar to PH menus or logon UDC's. The need to log on to the system is eliminated, and in some cases, hitting RETURN on an unopened monitor system device will not even display a colon (:). By educating users of the responsibility of the monitor programs (all transactions actually passing through the monitor program) they can become aware that their transactions may be put under scrutiny.

Prevention

Because users don't have to log on to the system, the capability to password session or monitor dialogue initiation is required to provide any prevention. Port security is at its ultimate level with either of these systems because of the system's reliance on device addresses for configuration. As with PH menus, identification of users by user-id, date, or time of day can be specified for additional logging.

Detection

As in PH menu systems, a supervisor program is responsible for either the session initiation, or the entire transaction-processing activity from a user's input. Reports of violations, or even strict procedural controls on transactions processed built within these systems can improve detection capabilities.

Integration

Some of the STARTSESS software has reached the same level of integration already available from PH menu systems. In the STARTSESS systems, applications are built into a device capability list, and passwords can be required by application. Monitor type systems are usually integrated into a single application, and are typically more limited in areas of application mixes.

Administration

Devices become the key considerations in configuring a STARTSESS or Monitor security system. The physical placement of users becomes the critical element in determining "who did what when." Dial-up lines are ruled out with either of these security systems because device allocation depends on an available terminal device. Although allocating these devices directly to communications equipment may be possible, the difficulty of implementation, and lack of definite user identity (location) probably rules out voice or rotary-line connections.

Summary

STARTSESS security methods offer all of the advantages available from PH menu systems, but are extremely limited where voice or rotary line connections are required. Monitor programs offer a very great deal of application and logon security, but tend to be tied to one application, and their user in a mixed application environment has never become prevalent. MPE user assignment could theoretically be eliminated with STARTSESS applications (allocate only one user per application if desired), and are necessarily eliminated in monitor security applications. By eliminating the large numbers of users allocated through MPE, one level of cross-referencing users (MPE users) for auditing purposes can be simplified.

A less obvious problem with both the STARTSESS and monitor security method is its reliance on physical factors sometimes beyond the control of data processing, and even user management...physical placement of users. EXAMPLE: If device 39 malfunctions (broken keyboard, no power, etc.), and the normal user for device 39 needs to use the system, how do we manage this situation? When do we return the "substitute device" back to its normal service? What if this happens on third shift, the last day of the month, and orders are being held from shipment waiting for device 39's user to make shipments? This dilemma could lead to the delegation of security-administration to the $4.50/Hr third shift operator. This could be a dangerous situation!! (awakening, perhaps).

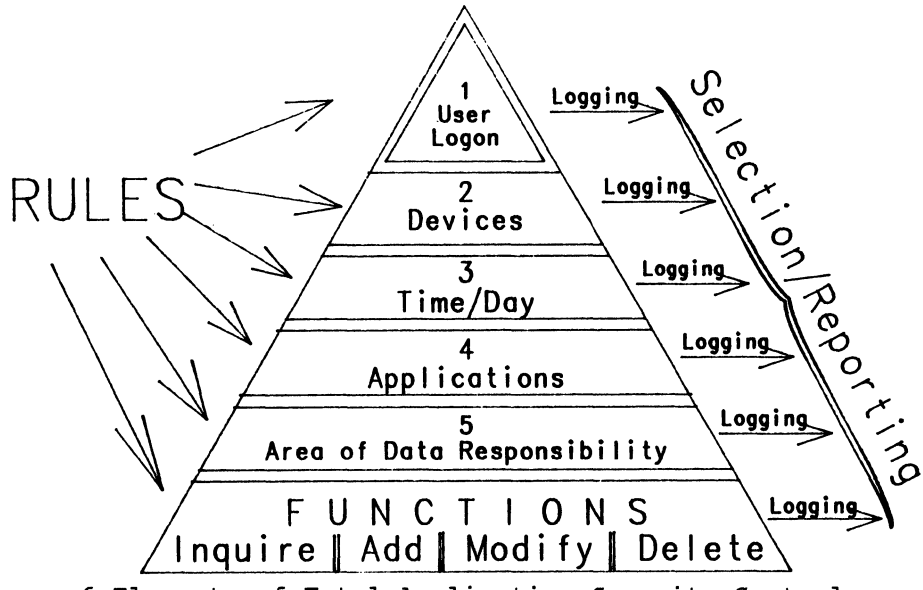
Future Solutions

In the pursuit of more secure computer systems, advances are made more rapidly where physical and operational security are concerned. Advances to aid the operational security for HP3000's are just around the corner. Encrypted passwords, encrypted store tapes, and accounting structure maintenance enhancements will probably be introduced by Hewlett-Packard or third-party vendors very soon. Application security and its relationship to logon security, however, have not been integrated, and may be the major challenge of securing and controlling systems into the 1990's.

The weaknesses from lack of integration between logon and applications cannot ultimately be "blamed" on the method of application-selection used. Instead, the standards for implementing functional security within applications is probably the leading culprit in causing an interface problem.

Total application security control, illustrated in figure 2, consists of six characteristics. Current security systems emphasize elements one through four. The area of data responsibility, or files maintainable by the user, are not considered in the general applications-security systems. The applications (AR, AP, Payroll, etc.) usually ask for another user-id or password to identify the user's capabilities in performing file-maintenance functions.

FIGURE 2



6 Elements of Total Application Security Control

Where double lines are shown, applicable restrictions can be applied. Rules establish the type and limitations of restrictions. Logging on all levels of the triangle, and selection from the logfiles provide reporting of all levels of access within this security environment.

The rules (to the left of the triangle), and the selection of logging records (to the right of the triangle), become the key items in the security system. Rules and logging ease administration, deterrence, prevention, detection, and recovery. The all-inclusive applications and logon security system must meet the following requirements:

1) Applications must be directly defined within the security system's rules.

2) Functions within applications must be defined within the security system's rules.

3) Access to these rules must be available to application programs via callable subroutines.

4) Access to logging routines must be available to applications via callable subroutines.

5) All parameters must be general enough to allow loose or tight restrictions as required by applications.

6) The security system should provide the guidelines that can be easily adopted in applications-development, building a security-integration methodology.

7) Access to this wealth of information on logons, applications, and capabilities must be secured from unauthorized access through encryption and procedural controls.

By solving more of the problems of the security administrator **BEYOND LOGON SECURITY**, a more comprehensive and secure applications environment can be nurtured for the HP3000 systems environment.

Bibliography

Books

Fisher, Royal P., Information Systems Security, New Jersey: Prentice-Hall, Inc., 1984.

Hsiao, Kerr, Madnick, Computer Security, New York: The Academic Press, 1979.

Parker, Donn B., Computer Security Management, Reston, VA: Reston Publishing Company, Inc., 1981.

U.S. Congress Senate, Committee on Government Operations, Computer Security In Federal Programs, Washington, DC: U.S. Government Printing Office, 1977.

Volokh, Green, Thoughts and Discourses on HP3000 Software, Los Angeles: VESOFT, Inc., Second Edition, 1986.

Articles

Berney, Karen, "The Cutting Edge", Nation's Business, April, 1986, pg. 57.

Blake, Isaac, "Computer Security and Legal Issues", INTEREX Detroit Conference Proceedings, October, 1986, Volume II, Paper 3315.

Engberg, Tony, "Reinforcing HP3000 Security", 1982 Second Annual Bay Area Regional Users Group Conference Proceedings, Also, Interact Magazine, January/February, 1983, PP. 38/43.

Firpo, Janine, "Security Concerns and Solutions", INTEREX Detroit Conference Proceedings, October, 1986, Volume II, Paper 3316, Also "Security: Solving MPE Pitfalls", Interact Magazine, August, 1986, pp. 56-60.

Hill, Peter R., " 'HELLO' - An unfriendly greeting, or an offer of seduction?", INTEREX Detroit Conference Proceedings, October, 1986, Volume II, Paper 3314.

Kunk, Joseph, "Utility offers G'day to system managers, users", The Chronicle, March, 1987, pp. 38-40.

LaDuca, Samuel W., "Security: What I Really Want", Interact Magazine, June 1985, pg. 55.

Lewis, Mike, "Computer Crime: Theft in Bits and Bytes", Nation's Business, February, 1985, pp. 57-58.

Peterson, I., "Federal Computer Security Concerns", Science News, October 12, 1985, pg. 230.

PROBLEM SOLVING IN AN HP3000 SHOP

Michel E. Kabay

JINBU CORPORATION
P. O. BOX 509 WESTMOUNT
MONTREAL, QUEBEC, CANADA
H3Z 2T6

INTRODUCTION

All of us have solved, are solving, and will solve problems in our work. To solve problems faster and train others to do so effectively, this paper presents a systematic approach to problem solving based on well-established scientific methodology. Examples are drawn from the HP3000 environment.

The paper treats the following topics:

- SET YOUR GOALS
- GET THE GLOBAL PICTURE
- DISTINGUISH OBSERVATION FROM ASSUMPTION
- DISTINGUISH OBSERVATION FROM HEARSAY
- DISTINGUISH OBSERVATION FROM HYPOTHESIS
- CHALLENGE YOUR HYPOTHESIS
- TRACK THE DETAILS
- RTFM
- RTFSSB
- GETTING THE MOST OUT OF P.I.C.S.
- HELP FROM YOUR LOCAL HP TEAM
- TIME AND MATERIALS FROM THE SEO

Copyright M.E.Kabay 1987

SET YOUR GOALS

Short-term objective of problem solving

FIX IT

Long-term objectives of problem solving

FIGURE OUT WHY IT HAPPENED

MAKE SURE IT DOESN'T HAPPEN AGAIN (IF POSSIBLE)

Copyright M.E.Kabay 1987

Find out what the person is trying to do:

PROG: How do I write octal data directly in the directory?

TECH: *Why would you want to do that?*

Because I have to alter the first 8 bytes of the file block, some parts of the file index block, and data in the group and account index blocks.

No, I mean, what are you trying to do? How would you describe the problem to someone who didn't know about the directory?

I have to change the name of the file.

How about :RENAME?

Oh yeah....

This principle applies equally well to non-technical issues:

Can I have system ALPHA for a production that lasts 3 hours and takes 2,000,000 sectors of space this weekend?

Bad answer: *Sorry, ALPHA's booked solid this weekend.*

Better answer: *Sorry, ALPHA's booked solid this weekend. But what's the problem--what do you need? Maybe we can solve it some other way.*

--and in this actual case, the programmer had erroneously assumed that only ALPHA would handle the job; in fact, system BETA was free and the special job ran perfectly on it.

Copyright M.E.Kabay 1987

DISTINGUISH OBSERVATION FROM ASSUMPTION

Case study:

The FORTRAN compiler doesn't work any more.

Oh? That's interesting; it hasn't had major modifications in years. What's the problem?

I have this program that was working fine, and now it doesn't work any more.

So when did you last run it?

Well, actually, I didn't exactly run it--it was another person.

Who?

Umm, you wouldn't know them:they're on another computer.

Same HP3000 model as ours?

No, actually, it's a CDC.

Say, how did you get this program, then? Tape? Diskette?

No, I typed it in from the listing.

ASSUMPTIONS:

(a) perfect transcription;

(b) identical versions of FORTRAN;

(c) identical implementation on different computers.

Copyright M.E.Kabay 1987

DISTINGUISH OBSERVATION FROM HEARSAY

Case study:

One of our terminals has double softkeys showing up.

What were you doing when it happened?

We were running program ABC.DEF in menu G when we hit the f3 key; and then the softkey labels appeared above the regular set.

What do the keys show? The same as the original set or different?

I don't remember.

Well, can you go see it again?

Actually, I didn't see it myself; C told me.

Gets C. [...C arrives...] C, tell me again what happened.

Well, I was running program ABC.DEG when...

Wait a minute, not program ABC.DEF?

No, that one runs fine. Why?

Copyright M.E.Kabay 1987

Case study:

There's some sort of problem with port 26 on system I--all we can get is an endless series of the letter 'k' in lower case all over the screen. It's hardwired, so it can't be a datacomm problem.

*No, actually, the **PROBLEM** is the 'k' all over the screen; we **DON'T KNOW YET** whether it's the*

__SCREEN__ or the

__KEYBOARD__ or the

__CABLE__ between the screen and the keyboard or the

__CABLE__ between the screen and the port or the

__PORT__ or the

__AIB__ micro-board or the

__AIB__ itself or the

__SIB__ or the

__IMB__ or the

__ATP__ driver or the main

__MEMORY__ boards or the

__CPU__ boards or the

__BACKPLANE__ connectors.

Copyright M.E.Kabay 1987

CHALLENGE YOUR HYPOTHESIS

Or "test your ideas".

So let's see this terminal.

IDEA: If it's the port,

TEST: then detaching the RS-232 connector will stop the 'k'--AHA, it doesn't. The garbage has nothing to do with the link to the computer. So we look at something else.

IDEA: Suppose it's the keyboard? Then

TEST: removing the keyboard connection will stop it. Yes indeed, it does. I see the connector's a littl bent... oh, I see:there's a pin out of place because perhaps someone forced the connector into place.

IDEA: If the pin is responsible, then

TEST: perhaps putting it back will fix the problem.

Yes.... it works now."

Copyright M.E.Kabay 1987

TRACK THE DETAILS

When a major problem occurs,

SIT DOWN AND RECONSTRUCT THE EVENTS

before trying out various solutions and workarounds.

Note exactly

WHAT you did

in **WHAT SEQUENCE** and

whether there were **UNUSUAL EVENTS**

before the problem

occurred.

DON'T PREJUDGE WHAT'S IMPORTANT

CUST: "We had a system failure XX a few minutes ago."

PICS: *"Exactly what did you do?"*

CUST: "The operator was working with spool files. He changed the outfence on ldev 6, then tried to alter the priority of a spoolfile on a remote spooled printer. Oh--he made a typing mistake on the classname: used MAX even though we had removed that classname a week ago. Then he--"

PICS: *"WAIT! That's it! The bad classname. There's a known bug that causes this system failure when you refer to a nonexistent classname. We'll send you a patch."*

Copyright M.E.Kabay 1987

R e a d t h e ... F i n e M a n u a l

Your *HP Reference Manuals* should be

- up to date
- accessible to programming and operations staff
- in numbered, labelled volumes
- cross indexed by key word in a separate list
- signed out to show where the volume is now
- returned within a few hours at most
- purchased separately for frequent users

Copyright M.E.Kabay 1987

R e a d t h e F i n e

S o f t w a r e S t a t u s B u l l e t i n

Every quarter, the ***SSB*** index should be scanned by the System Manager and all problems involving **SYSTEM FAILURES, HANGS,** and **DATA CORRUPTION** should be highlighted in colour.

< *and you learn*
<=== *a lot about*
< *MPE and subsystems*

Every fortnight, the ***update*** issue should be scanned and added to the binder holding the quarterly ***SSB*** and the biweekly update issues.

But there has to be more than one system for this to be useful since otherwise you can't get at the info.

> In some large shops,
> information about SFs
> and HANGS is added to
===> online **HELP** catalogs
> which make precise
> information available
> by failure number.

Copyright M.E.Kabay 1987

GETTING THE MOST OUT OF P.I.C.S.

Phone-In Consulting Service

Atlanta, Georgia
Santa Clara, California

1) Clarify the problem yourselves before calling PICS.

For this initial phase, don't worry about neat notes--just write down everything you remember about the events.

--What were we trying to do?
--How were we trying to do it?
--Exactly what did we observe?
--Then what did we do--in what order?
And what happened?

2) Now write down the details in chronological order and try to make sense out of them.

3) If it's a SYSTEM FAILURE,

a) check section IX of the *System Operation and Resource Management Reference Manual* (binder #2 of the standard Reference Manual set) to see what the SF means and what to do about starting the system again;

b) you will probably have to take a memory dump and possibly a shift-shring dump;

c) if necessary and possible, restart the system with a WARMSTART to save spoolfiles;

d) As the system comes up, find or remember your PICS *SYSTEM IDENTIFIER* and call PICS;

e) always =SHUTDOWN and start with a COOLSTART if permitted; otherwise get your KNOWN-GOOD COLDLOAD TAPE and COLDSTART or (gasp!) RELOAD.

Copyright M.E.Kabay 1987

4) Do your homework before calling PICS on a subsystem or application system problem

   a) Look in the Reference Manuals for a reasonable period (how about up to 30 minutes?) to be sure of the principles (how it's supposed to work);

   b) Ask colleagues for tips and explanations of the application software problem.

   c) Will avoid the ever-embarrassing, *"Look on page 3-13 of the Reference Manual"* answer to your question.

   d) Will also keep your reputation high with PICS staff and may help you get faster cooperation from them when you really have a problem (*"This lady really knows her stuff; if she's calling, it must be serious--I'll call her right away."*)

5) Look in the ***Software Status Bulletins*** when there are system failures or hangs while waiting for PICS to call back.

6) Get out your lists of

   a) **PATCHES;**

   b) **system software version numbers;**

   c) **PRIV MODE *NON-HP*** or **UNSUPPORTED UTILITIES** being **RUN;**

   d) **system configuration;**

   e) **previous SF** of this kind (in the same range).

7) Keep records of exactly **WHOM** you speak with **WHEN** about **WHAT** in dealing with PICS. Record the PICS-ID of each call prominently on all your records for each problem.

8) For large shops with several systems and many peripherals, consider a small database for appropriate information about all PICS calls and other technical problems. A microcomputer can be a good choice so you can find information even when your HP3000s are down.

Copyright M.E.Kabay 1987

HELP FROM YOUR LOCAL HP TEAM

WHAT PICS CANNOT DO

- PICS cannot debug application software from NON-HP sources.
- PICS cannot repair your software even if they find the problem.
- PICS can install only certain patches.
- PICS cannot physically repair your hardware.

BUT YOUR LOCAL SE AND CE CAN.

THEREFORE YOU SHOULD

- be on good terms with your local SE and CE;
- keep them up-to-date on your technical problems;
- be as precise in your dealings with local staff as with PICS staff;
- keep the managers at HP informed of problems and progress.

Copyright M.E.Kabay 1987

TIME AND MATERIALS FROM THE SEO

THE LOCAL SEO

- can help you debug your own applications on a TIME & MATERIALS basis (that means they will help you, but it's not part of your software support contract)

- will negociate special training courses for your staff at HP or in your offices

- can help set up benchmarks at reasonable cost

- have a great deal of experience on hand in a wide variety of situations

- have access to a worldwide network of human and technical resources to help you solve your problems

Copyright M.E.Kabay 1987

Supporting Remote Locations

by
Patrick J. Kelly
PACO Pumps Inc.
Oakland, CA

OUTLINE

I. Introduction
II. Application System Planning
  Data Flows
  Database Strategy
III. Processor and Data Communications
  Processor Options
  Data Communications Options
IV. Day-to-Day Support
  Equipment
  Computer Operations Scheduling
  People-to-People Communications and Troubleshooting
V. Training Remote Site Users
VI. Conclusion

I. INTRODUCTION

Today I am going to talk about supporting remote MANMAN/Mfg and MANMAN/OMAR users.

First, let me describe our company, PACO Pumps. Like most "smokestack" industries our company had not kept pace with computer technology. In 1983 we were batch processing our manufacturing and order entry software on an IBM mainframe which was located at the parent company (Baltimore Aircoil) headquarters in Baltimore, Maryland. The PACO headquarters in Oakland was a remote site to the Corporate Data Center. PACO's remote sites were served by mail.

Today, we have 90 terminals connected to an HP3000/70 which is located at the PACO headquarters in Oakland, CA. We have our own data communications network linking sites in 4 states to the HP3000. We run ASK's MANMAN/Mfg, OMAR, PLANMAN and will be installing MANMAN/Accounts Payable. PACO is very "remote-site-oriented" because much our our company's activity occurs outside the headquarters location.

We will try to answer three questions today:

1. What remote support techniques do we find successful?

2. How did we make tradeoff decisions?

3. What are some of the limitations of the ASK software and how did we overcome them?

II. Application System Planning

Data Flows

In planning systems design or implementation we have found it useful to develop a data flow diagram of the company's operations.

Figure 1A and 1B describes the material flows and order flows at PACO Pumps at a macro level. The headquarters and main plant is in Oakland, CA. Like an automobile dealership, a Branch sells new pumps, services pumps and sells repair parts, all for a local market. Some of the branches even rent pumps.

Database Strategy -- MANMAN/Mfg

We started our support of remotes by including them in our plans from the beginning. Before we bought the software we knew that MANMAN/Mfg and OMAR have a single-plant architecture. The architecture assumes a standalone cost/profit center, with its own warehouse, shop(s), bills of material, materials management group, etc.

We toyed with the idea of changing the software to handle all the locations in one database. We found what ASK has found: that there is no single multi-plant problem, so there is no single multi-plant solution. We did not expect much help from ASK for years. Our data and control flow pattern, figure 1, is certainly one of thousands of possible configurations that ASK might support.

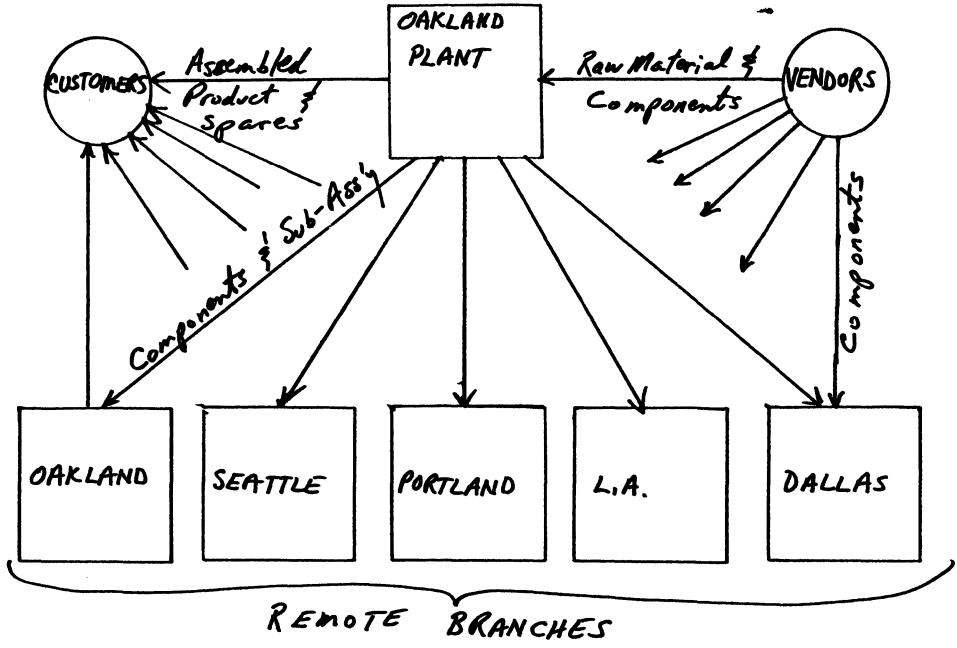
What we wanted to do was avoid all the source code changes, yet still serve multiple sites using a MANMAN/Mfg package that is single-site oriented. We analyzed every file in the MANMAN database, and every report and concluded that for a manufacturing site, there was little data actually shared between the sites. It seemed that a separate database per site was the way for us to go. Separate databases were also less risky. If we were wrong, it would be easier to put the separate databases together than it would be to split up a consolidated database.

We encourage our key users to participate in such decisions. We do this informally as well as at our regular, formal MIS Steering Committee meetings. It helps to keep MIS in synch with the direction of the business and keeps user management aware of where MIS is headed.

The major issue of concern to Accounting and Manufacturing was data integrity. The separate database, password controls and account/group structure was more than enough to ensure an acceptable level of data integrity.

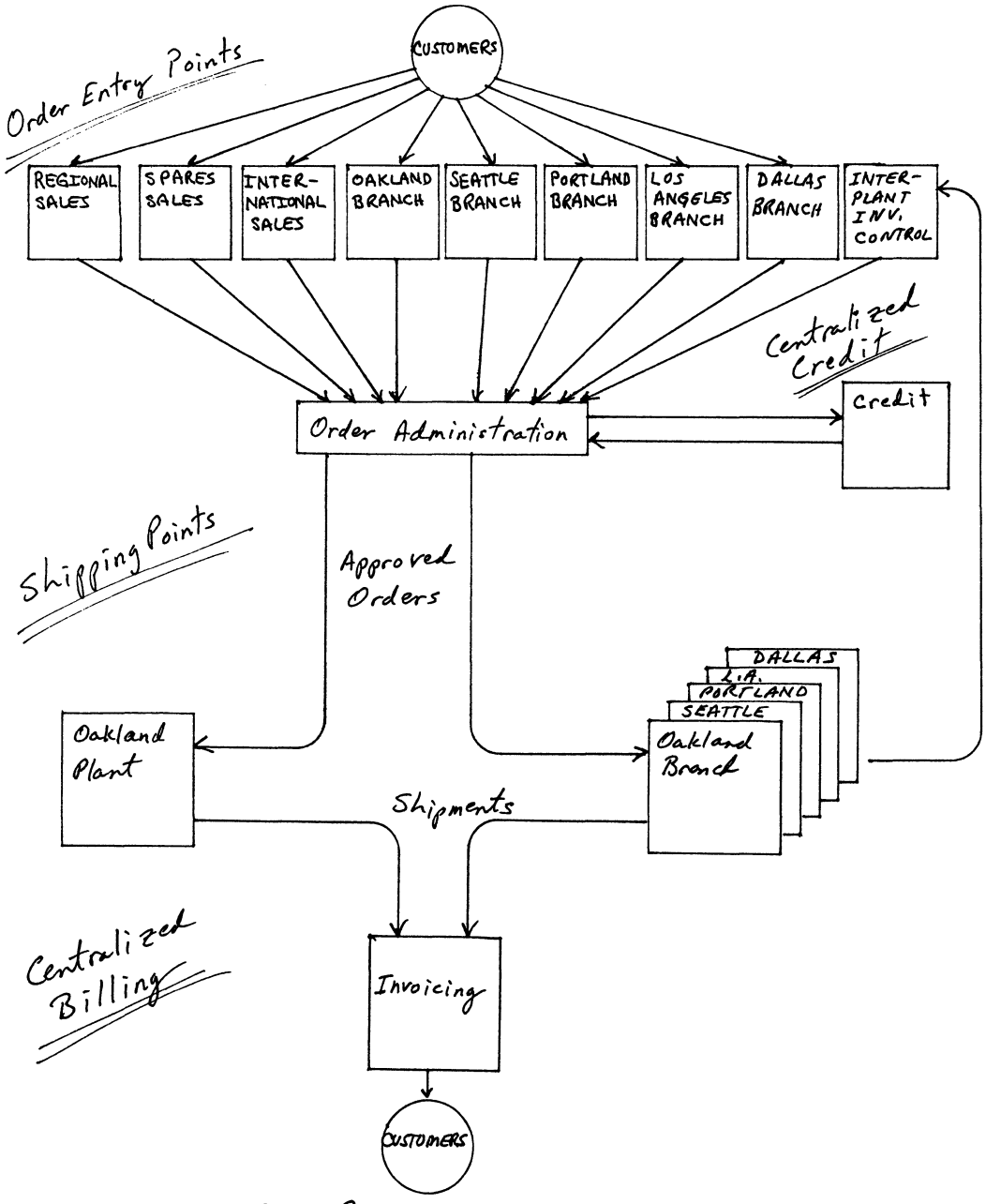
Inter-database Data Transfers

During these discussions we defined a few areas where we needed to develop software to automatically handle data transfers from one database to



MATERIAL FLOW

Figure 1A



Order Processing Flow

Figure 1B

another:

1. **Item masters** are added by Engineering only to the Oakland Plant database. The new item masters need to be entered in all 6 databases. This is done with a series of QUIZ programs that build batch jobs that do the updating.

2. **Material cost** at a branch is the Oakland Plant manufacturing standard cost marked up by a freight factor if the part is "bought" by the branch from the Oakland plant. Costs need to be kept in synch monthly.

3. **Inter-plant transfers** between the main plant in Oakland and the remote branches are handled in the following manner: a) the branch enters a purchase order into the branch database (to purchase material from the Oakland Plant); b) the branch user prints a copy of the purchase order in the Oakland inventory planning department, c) Inventory Planning enters the order to OMAR as a spares order. This process could be automated.

4. **Bill of Material** --- Our branches need access to the BOM's which are only stored in the Oakland Plant database. They use the LIST,2xx series on the Oakland Plant database with a password that allows them to do only list commands. Parts are miscellaneous-issued to the work order (actually a service order) in their local branch's database.

5. **Consolidated Reports** --- A few QUIZ programs have been found to be sufficient.

The approach we used suggests that a fast way to get multi-plant capabilities in MANMAN/Mfg is to set up multiple databases and then develop a few utilities.

Obviously, there is a penalty in terms of redundant storage of data. We are spending about four hundred dollars per month for lease payments on extra disc storage plus the cost of backing up and managing the disc space.

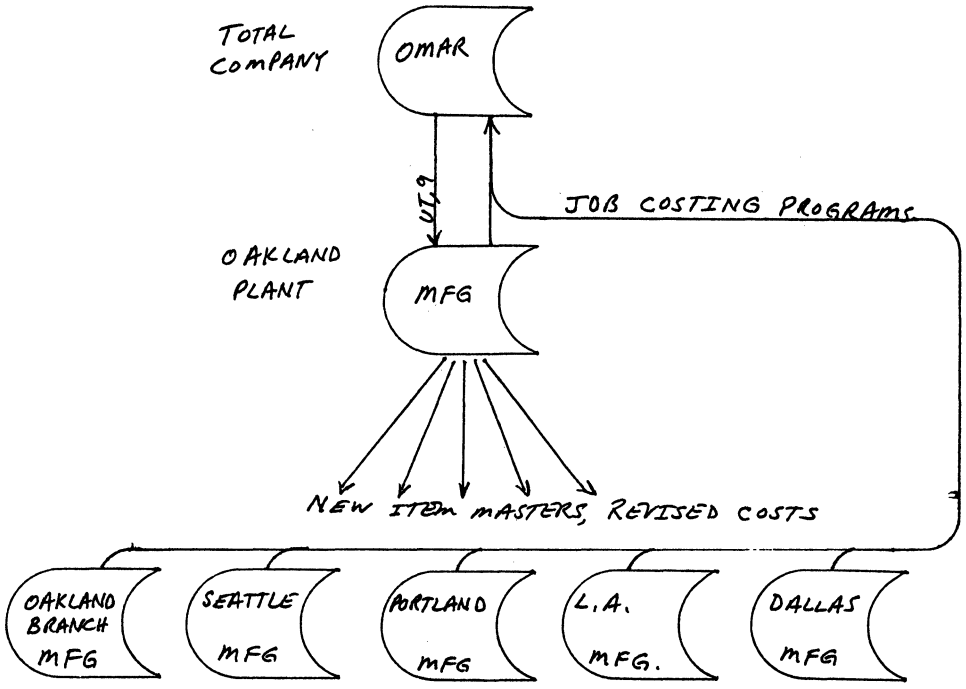
Data redundancy does have a performance advantage. By having multiple databases we have spread the company's total transaction load across several IMAGE databases, and avoided having one database be a bottleneck. This appears to be one of the reasons why we were able to run a large number (65) of terminals on an HP3000/58. At the time, we joked that had one of the biggest, (and slowest) HP3000 series 58's.

Of more long range value, we now have our company in a more flexible position for the future. Should ASK and HP offer a low priced MANMAN/Mfg + HP3000 package, then we could easily move our MANMAN/Mfg databases onto processors located at each branch.

Database Strategy -- MANMAN/Omar

Now that I have made the case for splitting databases, let me explain to you why we decided to implement one single OMAR database for all sites. We did not want to modify any source code, yet we had to satisfy these requirements:

1. **Accounts Receivable & Credit** --- centralized credit management function.



Database Configuration

Figure 2

2. **Consolidated Reporting** --------- job costing & reports by shipping location

3. **Orders Cross Location** ---------- half of all the customer orders are shipped from the Oakland plant MANMAN/Mfg database.

As you can see, the relatively neat separateness of manufacturing is just not present in Order Entry. The data flow diagram we talked about earlier confirms this.

We decided to put all the sites into one database. See figure 2. Our main motivation was to avoid modifying the ASK source code or database structure to accomodate the remote locations.

We established the single OMAR database.The OMAR Finished Goods Interface is tied only to the Oakland Plant database so that a T,210 shipment transaction relieves inventory in the Oakland plant MANMAN/Mfg database. This works fine for orders that ship from the Oakland plant. Branch shipments are processed with a T,210 against a non-nettable location called BRANCH (we use muliple location inventory). This creates some maintenance work to establish and zero-out these locations.

The shipping location is identified by the group code. If negative, then the shipping location is a branch. The negative code keeps the demand out of the Oakland Plant demand file and allows QUIZ reports by shipping location.

Inter-plant transfers from Oakland to the Branches are entered into OMAR against dummy customer, sales agent and product numbers. This keeps the statistics accurate. Again there are some maintenance jobs to maintain the files.

We addressed the need for data integrity by redundant coding. The order number, group code, note codes and rep number all uniquely identify which branch office originated an order. This makes it possible to trace "who did what" if data is inaccurately transacted. However, we are not completely satisfied with this level of data integrity. If all these fields were entered inaccurately, perhaps intentionally, then it would be difficult to trace who entered the data. Since there is no audit trail file, the system vulnerable to this kind of activity. At this time it seems that the only sure solutions involve modifying the source code.

III. Processor and Data Communications Systems Architecture

Processor Options

It would be very useful to our company to be able to provide a microcomputer to our small sales offices (PACO has 60), and small HP3000's to our major branch offices, all seamlessly connected via data communications to the central MANMAN/OMAR database in Oakland. Unfortunately not all of these options are affordable or supported by the ASK software. See the figure below:

Data Comm Options	Processor & Database Options: Multiple Micros Store & Forward	Distributed HP3000's with Local Databases	Centralized HP3000
Dial Up	NOT AVAILABLE (ASK Software restriction)	OK for Mfg, not for OMAR. Requires highly Skilled MIS and Users.	Simple Equipment Low Equipment Cost High Datacomm Svcs Cost ASK Software--no problems
Packet	NOT AVAILABLE (ASK Software)	Same as above.	Costs Difficult to predict
Leased	NOT AVAILABLE but why do it if you have a leased line?	Same as above.	Costs predictable Familiar technology Extra capacity for future applications
Satellite	NOT AVAILABLE	Same as above.	For very high capacity only. Too expensive for less than about 20 terminals. Concern about round-trip delay.

Processor & Data Communications System Architecture Options

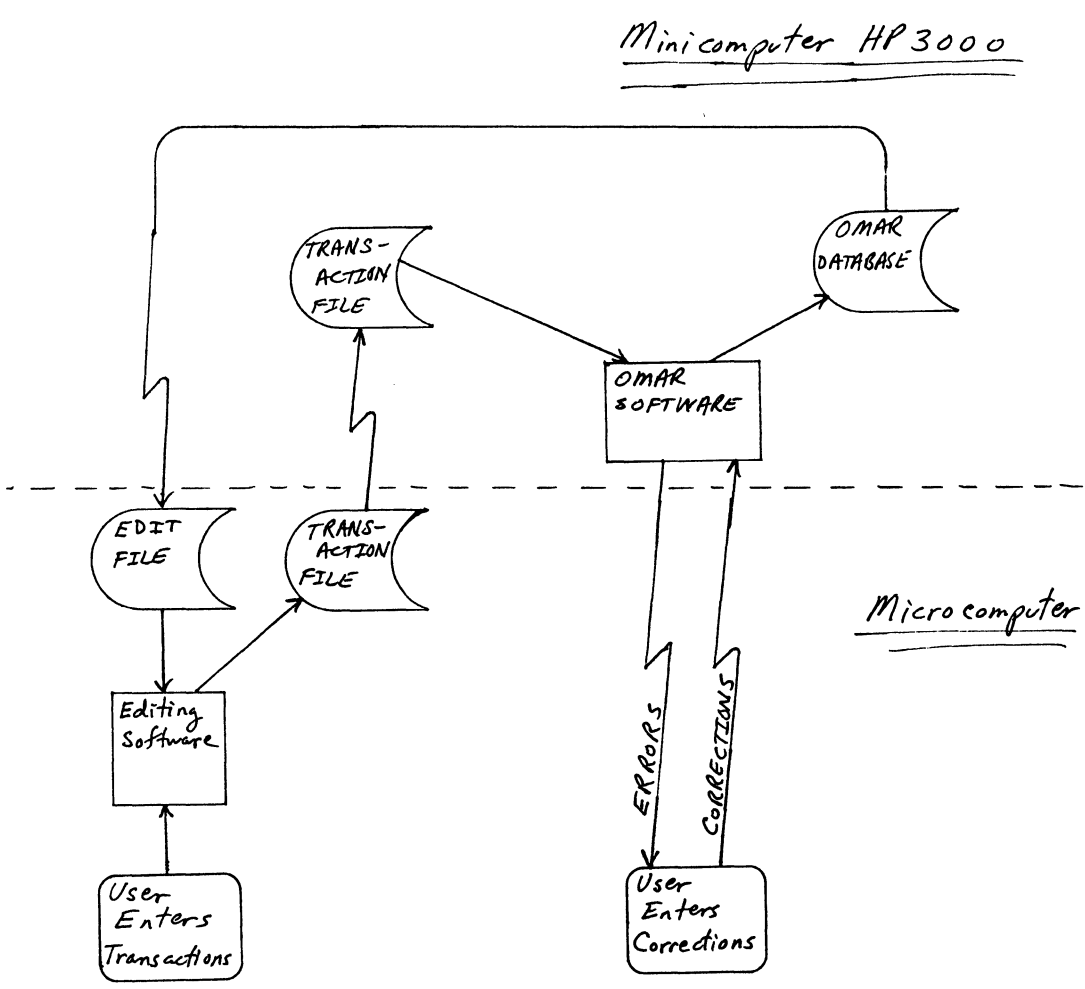
We would like to see the order input portions of OMAR support an "interactive batch" style of data input. See figure 3. This would allow us to use microcomputers more widely and simplify the data communications links.

Though these comments may sound like complaints against ASK and HP, they are the "complaints" of a happy customer who wants more. The single-processor architecture has been successful for us. We have smoothly upgraded from an HP3000 series 40 to 48 to 58 to 70 in a three year period. Now we are ready for something more complex.

Data Communications Options

In 1985, we decided to bring the remote sites onto the MANMAN/Mfg system. We tried some "cheap and dirty" modems first, but quickly replaced them with Paradyne FDX2400 error-correcting modems, which worked well. These supported one terminal and a slaved Thinkjet printer at each remote branch.

To implement OMAR we needed additional terminals per branch and a faster



"Interactive Batch"

Figure 3

printer. We investigated WATS, leased lines, packet networks, and satellites. We decided on a leased line network. See figure 4.

We chose DCA (Data Communications Associates) and CODEX for switching multiplexers and modems respectively. Mixing vendors' equipment has not been a problem.

Reliability was a major consideration because we did not want to expend resources troubleshooting. We bought every reliability-enhancing option available. Every CODEX modem has dial-backup capability. The DCA switching mux's have redundant power supplies and we stock spare mux boards in Oakland, just in case.

Our headquarters MIS department in Baltimore, MD plays a key role. They have a line monitoring system, called Data Network Control System (DNCS) by CODEX, that continuously scans each line and alerts the operator in Baltimore if a line begins to deteriorate. Dial backup can be initiated from either Oakland or Baltimore. The switchover to dial backup happens so quickly and smoothly that the branch users usually never notice that it happened. A similar system provided by DCA monitors mux's.

IV. DAY-TO-DAY SUPPORT

Level of Effort

It is generally understood that implementing a system at a remote site is more costly than implementing at the central site. It is not so apparent that the day-to-day support costs are also much higher. Due to data communications costs, the total equipment cost of a remote terminal is much higher than the cost of a locally wired terminal. The added complexity of the equipment requires a higher level of technical support. Different working hours make standard schedules impossible to adhere to. There is substantial economies of scale in a centralized computer center but it is less than it might appear.

Remote Site Equipment

We chose a Laserjet for the branches because we want to use the Quotation features of Rel. 6.0 of OMAR. See figure 5. All terminals are HP2392's, as are most of the terminals at the headquarters. This standardization has made it possible to easily train people, write standardized procedures, and it simplifies support over the phone.

Datacomm equipment is the same at each branch. The only difference is in the number of terminal ports.

All remote computer equipment is from HP and is covered by HP service contract administered from Oakland. We want avoid burdening the branches with arranging for maintenance of computer equipment. Their job is to sell pumps, ours is to maintain the equipment. See figure 5.

Operations Scheduling

When we first brought the MANMAN system in-house, we did not expect the system and the users to require so many batch jobs. By mid-1986 we were

manually scheduling over 2000 batch jobs/month and making many errors. We installed OCS (Operations Control System) and this greatly improved our error rate and batch processing capacity.

We also found that the operations group had to review thousands of $STDLIST output every day. It was impossible for human beings to read this output and catch all the errors. SMP automated this task for us. With OCS and SMP we can now run over 3000 jobs/month and with an error rate that is a fraction what it was with only 2000 jobs/month.

Our backup schedule also became a bottleneck. Besides the normal growth of the 6 MANMAN/Mfg databases and one OMAR database, the Oakland Plant expanded operations on the second shift which effectively doubled prime time. We tried pre-mounting a tape and initiating the STORE job by the operator from home at 5 a.m. This helped, for awhile.

A third shift operator was the best solution. Backups are now done between 12:30 a.m. and 2:30 a.m. We are now limited by the speed of the 7978 tape drive, as are all large HP3000 users.

Our normal schedule is shown in Figure 6. We have a 24 hour maintenance contract with HP. We schedule repairs and preventive maintenance on off-hours whenever possible.

People-to-People Communications with Remote Sites

We have a direct telephone numbers (DID) for everyone in the computer room. This makes it easy for MIS operations to call a branch and for the branch to call the Operations group.

Psychologists tell us that the telephone conveys 50% of the information of a face-to-face conversation. To maximize the quality of that 50%, we have cultivated at least one person at each site to be the "coordinator" with MIS. This person communicates computer availability changes and helps with troubleshooting and scheduling repairs. The coordinator also tends to be the funnel through which all enhancement requests pass through, though that was not intended.

The remote users are encouraged to call the HP3000 operator first. From our console log we find that 80% of users' problems can be resolved at the HP3000 console. Everyone in the department has some degree of day-to-day maintenance responsibility. This is defined on the troubleshooting tree diagram. See figure 7.

The branch managers and coordinators at the remote sites have a copy of this diagram so that they understand the process of problem resolution. We wanted the remote site users to never feel that their problem was a "hot potato" that was being passed from one individual to another. They now understand that the technology is complex and all problems cannot be resolved by one person. Referral of a difficult problem is something they expect to happen.

For example, there are two people in the "systems and programming group". One is responsible for MANMAN and one for OMAR. Each is a backup for the other. They take turns installing the new releases, both have been trained in both OMAR and MANMAN. They also have other responsibilities such as technical support, and applications development.

Just to keep everyone honest. The user is never required to follow the procedures outlined in the diagram. We only advise that we can usually take care of their problems faster if they follow the procedure. If a user habitually follows another path, then we examine why.

Disaster Protection

Disaster protection takes on a new importance with the addition of remote sites. For example, when only Oakland was on the computer we were not concerned about power failures, because the power to the users' terminals also failed. Now we have users in remote cities who could continue working. For short-duration outages, the remote sites experience double the downtime --- theirs and the central site's.

At present we exploring the alternative ways to cope with this and related disaster protection problems.

V. Training Remote Site Users

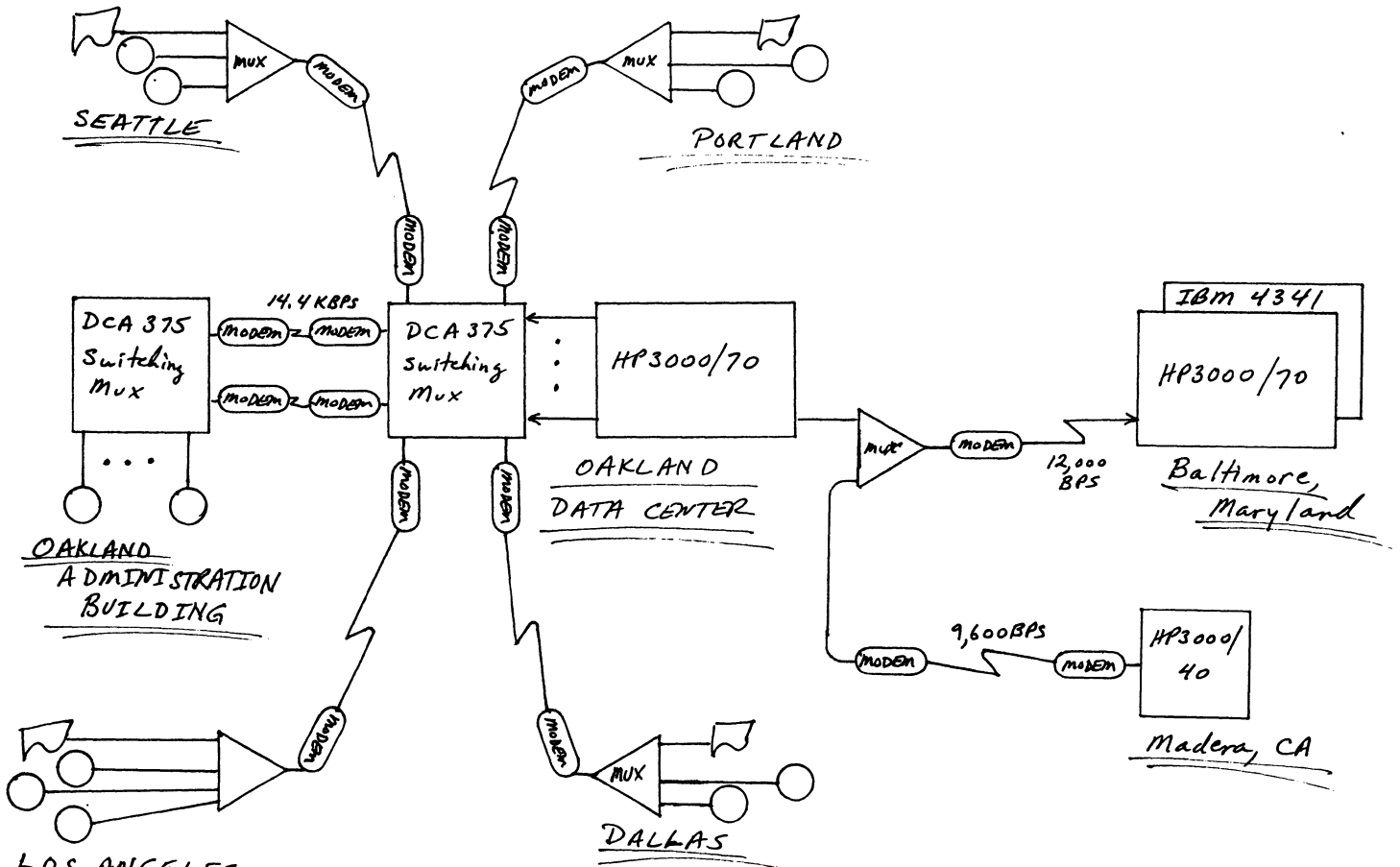
At one time or another, we have tried about every training method on our central and remote users --- outside courses, inside courses, one-on-one consulting, APICS courses, self-paced courses, video tape, and train-the-trainer.

For remote users, here are a few of the things that we found worked well:

1. Train remote users at the central site.
2. Have a central site user train the remote users in small groups.
3. Teach the remote user only what they need to know, so that when they go home they can immediately begin to use the system.
4. Break up your implementation into pilots. Do a pilot at the central site first and get the pilot participants involved in the training of the remote users.
5. Develop a condensed procedures manual based on the ASK manuals. It is their surrogate on-site expert when they return home.

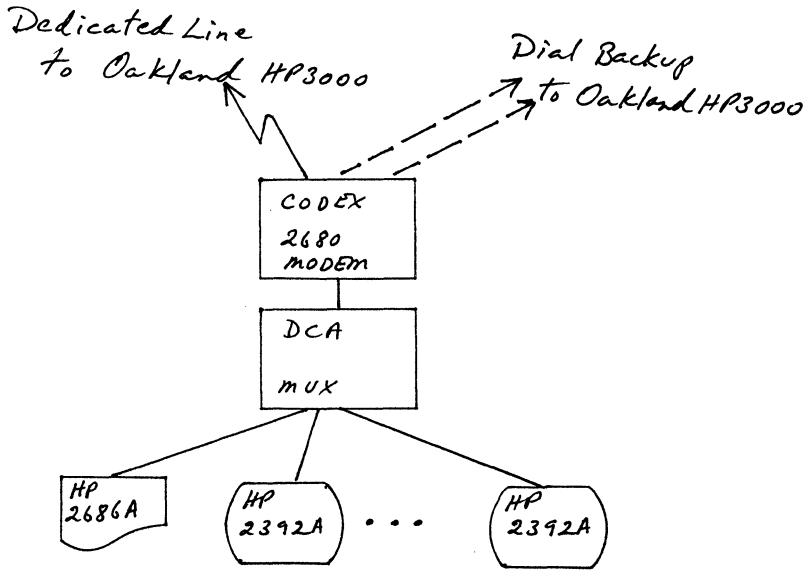
V. Conclusion

Almost every aspect of an MIS Department must change to accomodate remote sites. Today, I have tried to cover the areas that are most important. I hope that this presentation helps you with your remote sites.



Data Communications Configuration

Figure 4



Typical Branch Hardware Configuration

Equipment		Service
Branch Hardware	— Multiple CRT's	— HP Next Day
	— Printer (Spooled)	— HP Next Day
Data Communications	— Modem	— Codex Same Day
	— Mux	— DCA Same Day
	— Line	— Dial-backup
Oakland Data Ctn.	— HP3000/70	— HP 4 hour Response
	— CRT's & Printers	— Same as Branches

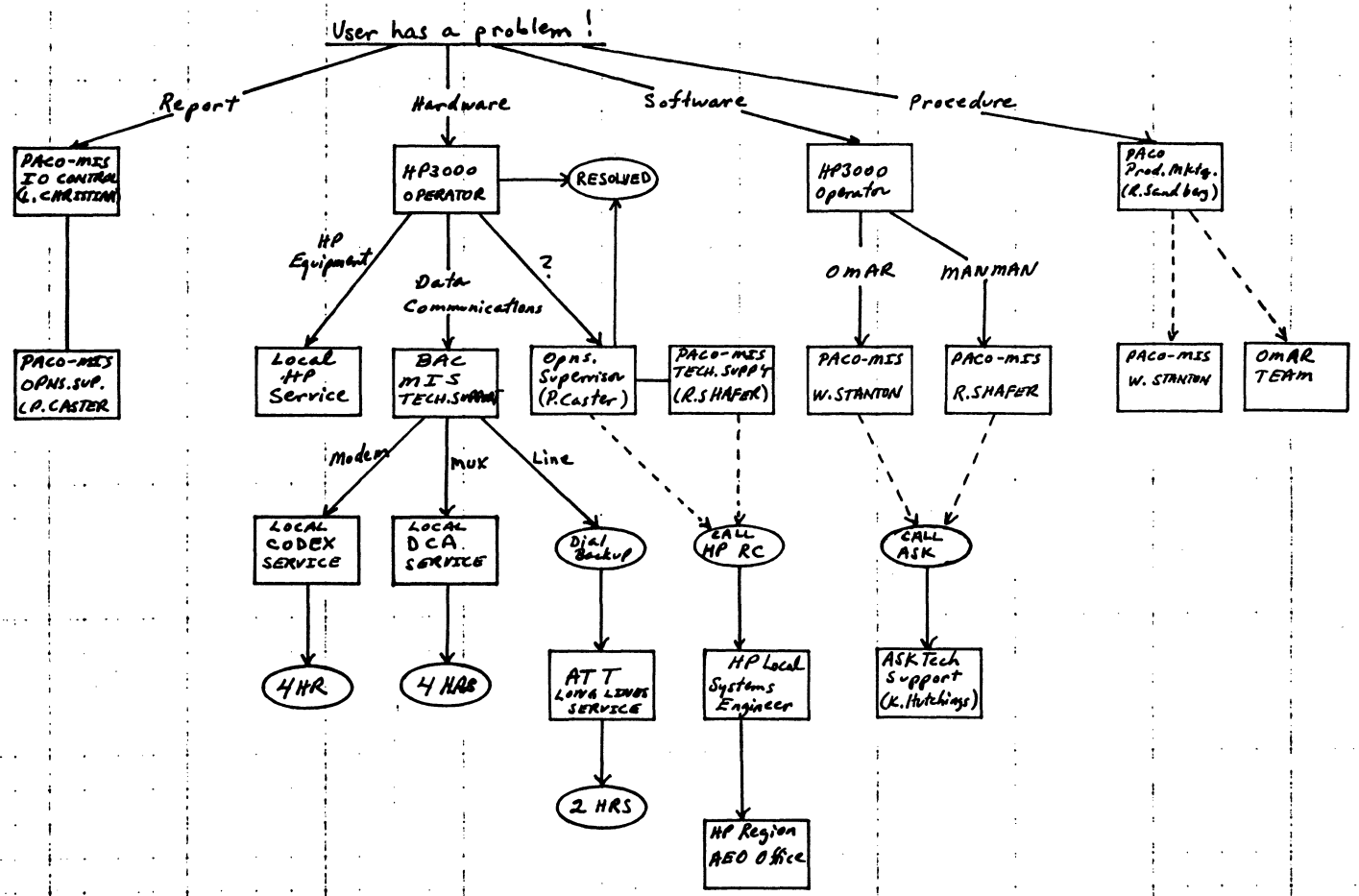
Figure 5

FILE:REMOTE.SUPPORT.FIG6,12/31/86

TIME OF DAY	HP3000 ACTIVITY	OPERATOR ON-DUTY	USER ACTIVITY
30	BACKUP	3RD SHIFT	END 2ND SHIFT IN
130	"	"	THE OAKLAND PLANT.
230	"	"	NO USER ACTIVITY.
330	BATCH PROCESSING	"	"
430	"	"	"
530	"	"	"
630	SYSTEM AVAILABLE	"	DALLAS STARTS.
730	FOR TERMINAL USERS	"	OAKLAND OFFICE.
830	"	1ST SHIFT	BRANCH OFFICES.
930	"	"	.
1030	"	"	.
1130	"	"	.
1230	"	"	.
1330	"	"	.
1430	"	"	DALLAS ENDS.
1530	"	"	OAKLAND OFFICE END
1630	"	"	BRANCH OFFICE END.
1730	" AND BATCH PROCESSING	NO OPERATOR	START 2ND SHIFT IN
1830	" "	"	THE OAKLAND PLANT
1930	" "	"	"
2030	" "	"	"
2130	" "	"	"
2230	" "	"	"
2330	" "	"	"

MONDAY THROUGH THURSDAY
COMPUTER OPERATIONS SCHEDULE

FIGURE 6



8/22/86
P. Kelly

Computer Problem Resolution Procedure Figure 7

AdvanceMail for the Portable PLUS: Its Design, Installation, and Usage

William L Kemper
Hewlett-Packard Company
Portable Computer Division
1000 NE Circle Blvd.
Corvallis, OR 97330

AdvanceMail Design Factors:

AdvanceMail for the Portable PLUS is an electronic mail application that ties the Portable PLUS to HPDeskManager (HPDesk), HP's electronic mail package for the HP 3000. AdvanceMail has been designed with the concept that a portable computer is not just a small computer, but is a computing device that can extend office functions far from the office. Rather than porting a desktop application to the Portable PLUS, the designers of AdvanceMail considered the intended environment and the users of the application. One of the first considerations was to create an application insulating the naive user from the complexities and difficulties of current-day PC data communications. The goal was to enable the user to switch between various data communications connections with ease and without needing to re-configure data communications parameters. Since data communications is not easily understood, the goal of the application was to provide the user with an easily changed set of configurations that could be set up for a specific HP 3000. Where a general office product is installed for a single connection, AdvanceMail had to have the ability to handle connections that included direct connections, modems, X.25 public data networks, and complex combinations of intermediary systems, e.g. data switches and security software, between the host computer and the Portable PLUS. The user had to have the ability to select a preset logon as easily as selecting a phone number.

Error detection and correction during data transmission becomes significantly more important in remote connections via modems than a terminal connection in the office. The application must be able to handle errors without requiring significant support. The user of the application will not have the ability to take corrective action. The application had to be able to recover following an error and not lose data.

In addition to dealing with connection management issues, the designers of AdvanceMail focussed on the dual nature of electronic mail, the creation and reading of mail versus the transfer operation. One of the advantages of a portable computer is the ability to take it away from the office. The creation and reading of mail could be done without establishing a connection to the HP 3000. The only time that a connection was needed was during an actual transfer operation. In this manner, the more time consuming tasks could be done without tying up a line. The transfers could then be done during off hours when phone costs were less and, by being batched, complete transfers in less time.

How AdvanceMail Works:

AdvanceMail provides the user with the ability to send and receive electronic mail messages with the Portable PLUS. AdvanceMail sends messages from the user to HPDesk over telephone lines or a direct computer connection, and receives messages to the user from HPDesk. HPDesk acts like a central post office, routing messages and files between different users.

The main features of AdvanceMail are:

- The **In Tray** contains messages received from other HPDesk users. They were received from HPDesk during previous transfers. Messages will stay in the In Tray until they are deleted by the user.

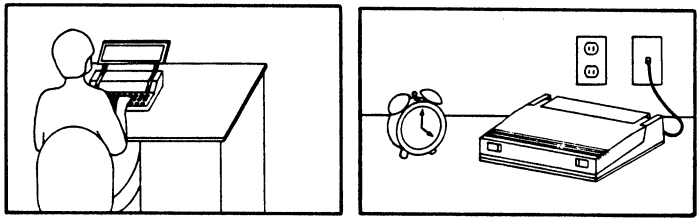
- The **Out Tray** contains messages which the user has written on the Portable PLUS and wants to send to others. They will be sent to HPDesk at the next transfer.

- The **Mail Room** lists all of the user's HPDesk messages, both those that AdvanceMail received during the last transfer and those that were not transferred. Messages may not be transferred due to user requests or errors during the transfer.

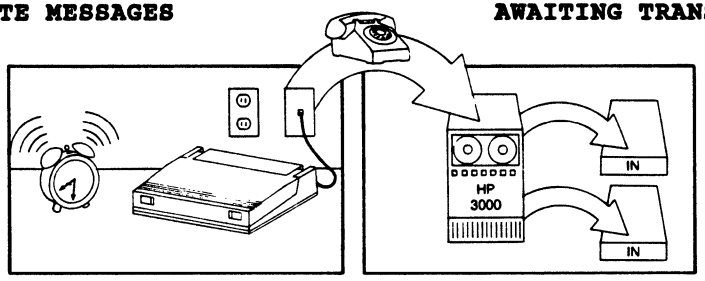
- Message transfers between AdvanceMail and HPDesk take place when the user specifies. It can be done while the user is away or sleeping. A transfer moves any messages waiting in the Out Tray to HPDesk for distribution to the people to whom the user addressed the messages. In addition, any messages waiting for the user in HPDesk are moved to the AdvanceMail In Tray to read later.

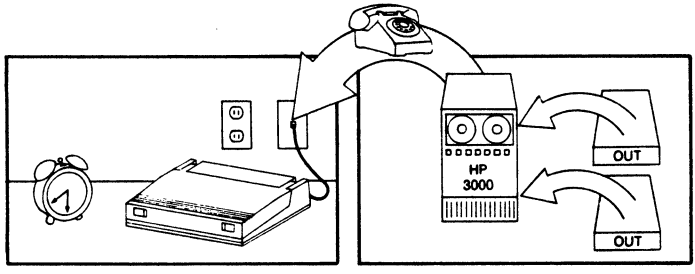
- The reading and creation of messages can be done offline when the user chooses. The actual transfers are batched which results in reduced charges. Prior to a message transfer, the user selects a preset configuration.

The following illustrations show how messages are moved from AdvanceMail to HPDesk and back.

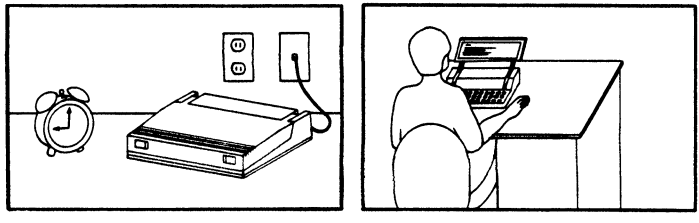


CREATE MESSAGES **AWAITING TRANSFER TIME**





TRANSFER OPERATION: SEND MESSAGES & THEN RECEIVE MESSAGES



NEXT MORNING **READ MAIL**

AdvanceMail System Requirements:

AdvanceMail has been designed for the Portable PLUS which is HP's totally battery powered portable computer. The Portable PLUS must have a minimum of 384K RAM, a software drawer, AdvanceMail ROMs, REFLECTION 1[TM] application ROM and a datacom connection. The datacom connection can be either a modem to connect to the HP 3000 by telephone or a direct connection to the HP 3000 via a serial cable.

AdvanceMail Installation:

The installation of AdvanceMail requires site-specific customization. The same work is required whether there are 1000 users or just one. In order for AdvanceMail to automatically transfer messages back and forth with HPDesk, it must be customized to fit the particular HP 3000 configuration (dial-in procedures, intermediary data switches, security programs, and so on). The customization is done in command and configuration files, which are then distributed to the users of AdvanceMail. After these files are created AdvanceMail can then be installed on the user's machine. The general steps that need to be taken are:

- Determine the data communications environment.
- Modify command and configuration files to match the data communications environment, i.e. determine what responses are needed by the hardware and software to logon to the host computer.
- Test the command and configuration files by trying an automatic logon to the HP 3000.
- Test file transfer procedures.
- Distribute pairs of command and configuration files to the users.
- Install AdvanceMail on the Portable PLUS with the customized command and configuration files.
- User configures Portable PLUS for individual HPDesk user passwords and phone numbers.

While the installation of AdvanceMail is not a trivial process, it is an easy product for end users to use, once it has been installed. The installation procedures are documented in "Setting Up AdvanceMail for the System Administrator." This manual assumes that the person setting up AdvanceMail, the system administrator, is used to working

with computers and is familiar with often-used computer terms. While familiarity with the Portable PLUS is not required, the system administrator should be familar with personal computers, especially MS[R]-DOS file structure and a word processor. Ideally the system administrator should have knowledge about the HP 3000, data communications, the Portable PLUS, and REFLECTION command language. Access to expertise in any missing areas should suffice.

The first step is to determine the data communication enviroment that AdvanceMail will use. This information will enable the system administrator to create command files that meet the needs of the environment. Some items to consider are:

- What type of connection is between the Portable PLUS and the HP 3000? (direct or modem)
- Are there intermediary systems involved between the Portable PLUS and the HP 3000, such as data switches or call back units?
- Is a logon security program in place with which the Portable PLUS must know how to interact?
- What phone number must be dialed to access the HP 3000?
- What responses must be sent to the host?
- What are the lengths of time delays between prompts?
- What account structure will users have? Must they enter name and account only, are there passwords with them, do they have to enter a group name and password, do they have to identify themselves to the system in other ways?
- Will the users be using a public data network, such as Telenet[TM] or Tymnet[TM]?

Much of this information can be determined by taking notes of an actual logon dialog with the HP 3000. This is done by using the Portable PLUS with REFLECTION 1 using the connection that will be used by the users.

After determining the logon procedures for a particular connection, the system administrator can select sections of sample command files supplied with AdvanceMail on a supplemental disc. Using a word processor or EDLIN, the system administrator can edit these command files to create a custom command file for each datacom connection. A separate command file will be used for each connection to be

installed on the user's Portable PLUS, e.g. direct at the office, modem at home, and modem using X.25. For each command file, there will be a specific configuration file that sets the data communications parameters, such as type of connection, Baud rate, stop bits, etc., that is called by REFLECTION 1 when a session is initiated.

The created command and configuration files are then stored on the Portable PLUS for testing. The testing is done in two phases, a logon test and then a file transfer test. The first test verifies the unattended logon process. By doing the test in two phases, any installation problems can be identified more easily. The logon test will simulate AdvanceMail calling REFLECTION 1 with a command file and other user parameters, such as accounts and phone numbers. These parameters will be passed to REFLECTION 1 by AdvanceMail in actual use. A successful logon will need to be terminated manually. The testing may require editing the command files several times to get timing factors incorporated. This process is repeated for each command and configuration file pair. The next phase can then be tested.

In the second phase of testing, logon descriptions are entered into AdvanceMail and activated. The logon descriptions are the user-specific pieces of information that are required to make a connection, e.g. phone numbers, user accounts, and user passwords. The Transfer Mail part of AdvanceMail will be executed. If it is successful, a summary screen will be displayed. Additional modification of the command and configuration file pairs may be required.

Depending on the AdvanceMail installation, the user may be instructed in installing the application or provided with a Portable PLUS with the necessary installation completed. To use AdvanceMail, the user will need to become familiar with the Portable PLUS first. After becoming familiar with the Portable PLUS, the user can quickly start using AdvanceMail to send and receive messages.

Additional AdvanceMail Features:

The sending and receiving of messages is the basis for AdvanceMail. In addition to these capabilities, other features include:

- Filters to specify actions to be taken on messages with matching patterns. Patterns can be matched in both the Sender and Subject fields.

- Automatic transfer of files from HPDesk to designated MS-DOS files for use by other applications, such as 1-2-3[R] from Lotus[R].

- Automatic printing and deletion of messages on the host.

- Creation and use of local distribution lists.

- Ability to reply to and forward messages in the In Tray.

- Unattended transfer at user specified time.

- Error log and recovery instructions.

In addition to these application features, the AdvanceMail transfer mechanism can be used to automatically transfer information without user intervention. As part of the installation procedures, the file transfer process can be changed to:

- Automatically receive messages meeting a predetermined filter criteria, such as subject or sender.

- Automatically send files without requiring the user to send them, e.g. a salesman's call report stored as an MS-DOS file.

- Add processing to be done before and after transfer or call other programs to run on the HP 3000.

All of these capabilities require additional customization prior to installing AdvanceMail on the user's machine, but extend AdvanceMail far beyond remotely accessing HPDesk.

An example situation may better explain the preceding capabilities. A sales force for a chemical distributor has been outfitted with portable computers. During the day each sales representative uses the Portable PLUS to record sales call activities on a template. When the sales representatives get home at night, they compose additional messages using AdvanceMail. Prior to going to bed, they connect the phone line to the Portable PLUS and set a transfer time for the Portable PLUS to wake up and initiate the transfer. At this transfer time prior to logging on to the HP 3000, special preprocessing will occur unbeknownst to the user. An MS-DOS program will be executed that appends a time and date stamp to the sales call report. A permanent send specification has been stored during the installation of AdvanceMail that will always send the sales call report file to the sales representative's manager. The sales representative does not have to mail the sales call report

each day. A permanent receive specification has also been created that will automatically update the price list spreadsheet on the Portable PLUS. If a message with "PRICE UPDATE" in the subject field is found in the HPDesk Mailroom, this message is automatically received and stored in the MS-DOS file "PRICE.WKS" for reading by 1-2-3. These advanced capabilities provide a mechanism for controlling file transfers and automatically performing tasks that could be forgotten.

Conclusion

AdvanceMail does require customization prior to the actual usage of the application by the intended user. This application is different from most personal computer applications which can typically be installed and used immediately. If data communications were simpler and all computer installations were the same, this customization would be unnecessary. Rather than assuming that all installations would be the same, AdvanceMail has been designed to be tailored to specific environments. The installation process is critical to the usability of AdvanceMail by the intended users. Once AdvanceMail has been successfully installed, it can be readily used. AdvanceMail extends HPDesk from the office by providing offline capabilities wherever the user wants, be it in the home or at 30,000 feet in an airplane flying from San Francisco to Houston.

MS[R]-DOS is a U.S. registered trademark of Microsoft, Inc.

REFLECTION 1[TM] is a U.S. trademark of Walker Richer & Quinn, Inc.

1-2-3[R] and Lotus[R] are U.S. registered trademarks of Lotus Development Corporation.

Telenet[TM] is a trademark of GTE Telenet Communications Corp.

Tymnet[TM] is trademark of Tymshare, Inc.

DISASTER RECOVERY - IF THIS HAD BEEN A REAL EMERGENCY..."

MITCHELL KLEIMAN
DIRECTOR OF COMPUTER TECHNOLOGIES
JOHNSTOWN CONSOLIDATED CAPITAL
2000 POWELL STREET
EMERYVILLE, CA 94608

Introduction

The title of this paper hopefully reminds you of a voice you heard from a radio or television set during a test of the emergency broadcast system, telling you that if this had been a real emergency, you would have received instructions. These words have been ingrained in my memory patterns as year in and year out at some point, on one or more radio and tv stations I heard that test. I have to admit that I listen to those tests carefully, waiting to see how long they wait, did it work correctly; and several times (mostly 10 or more years ago) I could have sworn I caught a problem or discrepancy or it took them too long after the long warning tone to put the voice on. But lately they've been doing very well. Those several years of practice seem to have paid off - the system appears to run smoothly.

I do not know many MIS departments that have adequate emergency plans in place, fewer that have tested them more than once, and fewer still that have instituted a regular testing of their emergency plans, insuring that their procedures, equipment, and trained personnel are up-to-date.

This paper addresses the basic needs and issues of building a disaster recovery plan for a company. A no-nonsense approach to reviewing needs, arranging for a hot site, preparing for using a cold site, and testing the plan.

This paper will also touch on related issues, such as pc backup methods, coordinating with system users to insure that the recovery plan recovers the users as well as the computers, insurance concerns, what are the real costs and benefits of preparedness, minimizing the administrative work of keeping the plan accurate, and how far to take your plan.

Outline

I. Why we call it an Emergency Recovery Plan and not a Disaster Recovery Plan.

II. Levels of Emergency

III. What do you Need, Really?

IV. HOT SITE - if you're going, who's going with you, what are you really going to need, and for how long are you going to be there

V. Cold Site - how cold should it be

VI. The Approach - where do we start, in what direction do we head, and how do we measure progress

VII. Testing the Plan - the first time, the second time, and the third time, and the fourth time

VIII. PC Backup - Who's responsible, with what, where, how

IX. Once the computer is up, who (and how are they) going to use it

X. Insurance - for what, how much

XI. Costs and Benefits

XII. Okay, now what do I do with it

I. Why we call it an Emergency Recovery Plan and not a Disaster Recovery Plan.

emergency - 1. an unforeseen combination of circumstances or the resulting state that calls for immediate action 2. an urgent need for assistance or relief

disaster - 1. a sudden calamitous event bringing great damage, loss, or destruction 2. an unforeseen, ruinous, and often sudden misfortune that happens either through lack of foresight or through some hostile external agency

Probably you will need to deal with emergencies more often than destructive disasters.

IF YOU DO NOT PREPARE FOR THE EMERGENCIES, THEY OFTEN TURN INTO DISASTERS. PREPARE FOR ALL TYPES OF EMERGENCIES.

II. Levels of Emergency

1. Applications Failure
2. Processing Interruption
3. System Failure or "System Hang"
4. Environment Failure
5. Hardware Failure (under 24 hours)
6. Hardware Failure (over 24 hours)
7. Computer Room Disaster
8. Building Disaster
9. Local Disaster
10. Regional/Statewide Disaster

III. What do you need, really?

Are you running a "big shop", a "small shop", or a Micro 3000?

Are you located below a restaurant, above the boiler, between a munitions factory and a nuclear reactor, on land-fill, in an earthquake prone region of the country?

Is your company dependent on the computers being up all the time, some of the time, or can they catch up after a week or two of down time?

What have been the (major) sources of down time in the past three years?

ASSUME NOTHING, ASK QUESTIONS, LOOK AROUND, KEEP TRACK OF INTERRUPTIONS.

IV. HOT SITE

a site separate from your computer room, with the appropriate environment (power, a/c, humidity); with a computer, associated peripherals, and terminals; immediately available for your use.

1. **What do you get for your money?**
   phone lines
   office space
   support staff
   living quarters
   Who else is going to be there

2. **If you're going --**
   when
   how
   who is going with you
   what are you going to need
   how long are you going for

V. COLD SITE

a site you could put a computer in

it may or may not already be prepared -
- **power**
- **A/C**
- **wiring for terminals**

Do you need one?
How long will it take to make it ready?
How long will you be there?
How much does it cost?
Where will the computer come from?
How will people use the computer there - where will they be?

VI. The Approach

1. Prepare for the worst case first, then work your way down to the simpler stuff.

2. At the start, determine:

   what is critical - which people, what information, what's the priority

   what is really needed - a combination of what you think you know, what people tell you, and what you should be prepared for, anyway

3. Document what you have first.

   If you have an emergency during the preparation of the plan, at least you'll know what you have, what you're working with, and what you need to replace.

   Then document what you would replace it with if you could/had to.

4. Research the topic of disaster preparedness. Read everything you can get your hands on, talk to the auditors (they like to hear that you are working on stuff like this and they have pages of checklists, etc. they would appreciate your filling out), and talk to your insurance people.

5. Start preparing the other areas of the company to do their bit for emergency preparedness. It will be of no use to the company if there is an emergency and you get the computer up and running but it cannot be used because there is no user procedure manual or no one arranged for the mail to be forwarded to the recovery site location.

VII. Testing the Plan

If you think it's going to work the first time, you are in big trouble.

Most sites have trouble keeping things working at their regular place of business, under normal circumstances.

After completing the plan, I suggest three tests.

Test 1 - test the procedures, walk through them, have manuals and programmers and operations personnel and technical support ready. Be prepared for failures, stupid errors and misunderstandings, and simple problems (eg. memory configuration problems, tape failures, nonexistent printer definitions, etc.). Expect everything to take longer than you thought. Take extra people along. Take notes. Take as many people as you can and your most trained personnel. Load your software, bring up all or part of your system. Print off one or two reports. Log on to several ports.

The purpose of test 1 is to identify all the things you haven't planned for and to give you a sense of the problems you will encounter in a real emergency.

After test 1, update your plan from the notes you've taken, retrain the people involved and do test 2.

Test 2 - this time send fewer and less trained people - for example, have a trained operator follow the instructions with the system manager observing and taking notes and timing everything.

The idea is for test 2 to uncover what is needed beyond basic operations training and the procedures of the emergency plan to provide a successful recovery. This type of test better insures that things are not taken for granted and provides training for all involved. Include in this test remote user dialing into the backup site. Note access problems, response time, time it took to prepare the system.

After updating the plan with notes from test 2, you should be ready for...

Test 3 - do as real a run through as you can of a sudden emergency, test how you would recover lost data, bring real users with you, how and where will they work, where do they get lunch from, what do they do while you are bringing up the system.

At this stage you should be working out the detailed logistics of coordinating with the other parts of the company, user documentation, data recovery procedures, critical tools needed to run the company, who are the critical people.

Repeat test 3 periodically (at least once a year). Make it interesting, try surprises, typical obstacles, and not so typical obstacles.

Always have someone record what happens, timing, problems, resolutions; and incorporate these findings into the plan.

VIII. PC Backup

What pc's?

Do you need to back up the pc's in your company - do any of them have critical corporate information?

What would happen if you did not backup the pc's?
What would be the impact if none of the pc's in the company worked?

Options:

1. **They back them up.**
2. **You back them up.**
3. **They don't get backed up.**

What is appropriate for your company?

Types of pc backups currently available:

1. **diskette backup**
2. **Bernoulli cartridges**
3. **tape backup**
4. **backup to a mini or mainframe**
5. **double hard discs**
6. **optical disk storage**
7. **paper copy**

IX. Once the computer is up, who (and how are they) going to use it?

are you organized so that lost information can be reproduced
how do you work around lost information

what are the basic transactions in your company, how will those transactions be handled at a remote site, what can you do to support that work

Also, what should you be doing once you get the hot site working?

How do you get to the cold site or home?
....and how do you get home from the cold site

what are the types of problems you are likely to encounter?
for your company, your equipment, your personnel

VERY OFTEN, WORKING FROM A REMOTE SITE IS COMPLEX.

DON'T WING IT.

THIS MUST BE PLANNED OUT IN DETAIL.

WHILE YOU'RE RUNNING AT THE HOT SITE, YOU PROBABLY WILL ALSO BE COORDINATING MOVING TO A COLD SITE OR RE-ESTABLISHING YOUR ORIGINAL SITE. THEREFORE THERE WILL BE TWO MAJOR TASKS EACH ONE REQUIRING YOUR FULL ATTENTION.

Split your staff into teams, once your system is recovered.

If you wait too long, you will pay big bucks($) for not planning. Remember the lead times involved.

X. Insurance

Find out what type of insurance coverage your company has.

Critical information:
- **cost on insurance**
- **what is covered**
- **how much are you covered for**
- **what are the appropriate procedures to follow when an emergency does occur, and what is the timing involved**
- **will your preparedness lower your insurance costs**

XI. Costs and Benefits

Costs:

time - to draft a plan, research and prepare information

money - hot site and cold site contracts, off-site testing expenses

Benefits:

increased knowledge of the company

increased technical training - recovery

working closer with more of the company, opportunity to see how critical work is done and possible impact on that you can have

with improved planning and procedures, bringing on new employees is made easier and they become productive faster

decreased stress level from worrying about, what if....

C.Y.A.

possibly lower insurance rates

keeps auditors busy and off your back

XII. Okay, now what do I do with it

Update it regularly.

Test it regularly.

Train more and different types of people.

Check on other people's documentation.

Make the tests harder.

Change the way the company operates if benefits are possible.

Talk about it, share ideas, incorporate new methods, automate the procedures and processes.

Thinking about using C...? (and if not, why not?)

Monika Khushf
Tymlabs Corporation
211 E. 7th Street
Austin, Texas 78701

The C programming language has become the belle of the computer languages ball. Its unusual and productive mixture of high-level structured programming statements and lower-level "close to the machine" constructs has been a key factor in its widespread acceptance by both universities and businesses. C's popularity is remarkable indeed: each month, new books describing the language appear; magazines such as Computer Language and BYTE devote almost entire issues to C; and several newsletters extol the language's virtues. Many universities now offer C courses, and increasing numbers of students are choosing it for their software projects. Major corporations and small companies alike are writing their new software (and rewriting their old) in C.

There are very few things the different computer communities — IBM, DEC, AT&T, Honeywell, Cray, Unisys, etc. — can agree on. One of them is C. They all love it.

THE HP3000 COMMUNITY RESISTS

All, that is, except the HP3000 community. At the 1986 Detroit Interex conference, turnout was low for the C talks. Several of those who showed up for the other language-related presentations were noticeably anti-C, and could be heard grumbling among themselves: "C is hard to understand"; "C is easy to abuse"; "C is just plain ugly."

To be fair, there is opposition to C on all machines, but the HP3000 community appears to be the toughest nut to crack.

WHY IS THIS SO?

Much of the naysaying can be traced to C's late arrival in the HP3000 world. This tardiness was in turn caused by the challenges inherent in implementing C on the HP 3000, from the compiler writer's point of view. Unsigned long arithmetic, for example, is not easily accommodated on the system. The Segmenter is openly hostile to many C features, including auto-initialization of global function pointers and character pointers to strings, and global definitions scattered among many files. Combine these difficulties with the limited memory the HP3000 has to offer (excluding the use of extra data segments), and the picture is, to say the least, discouraging.

While C was therefore unavailable on the HP 3000 for quite some time, there were several other perfectly acceptable programming languages to choose from. Each offered some of the advantages associated with C, as well as attractive characteristics of its own. C on the 3000 is the new kid on the block,

and programmers who have been coding in PASCAL or FORTRAN or SPL for years are understandably reluctant to change now.

C'S UNIQUE PURPOSE

However, there are persuasive reasons to consider such a change. C was designed to fulfill a particular need: the need for portability. C offers the highest degree of portability of any language, coupled with the highest degree of machine interaction possible in a portable language. And the ANSI standard committee, which includes representatives from all the major computer manufacturers, is committed to assuring that as the feature set and flexibility of the C language expand, they do so with no loss in portability.

WHY PROGRAMMERS LOVE C

C has qualities programmers love - once they get over their initial fear. It has a clever preprocessor, which not only handles compiler controls such as #include and #if, but also expands macros with arguments, concatenates string literals, and "stringizes" arguments. It has a powerful run-time library. Its high-level features include structures, unions, string constants, pointers to functions, and type names. Its bit-manipulation, cast, and address-of operators can make low-level access to such things as absolute addresses and pieces of words a snap. And the prefix and postfix operators not only alleviate the programmer's increment and decrement burdens (while (array[i] == ' ') i = i + 1; becomes while (array[i++] == ' ');), they enable the compiler-writer to generate more efficient code.

WHY MANAGERS LOVE C

C has qualities managers love, too. The commitment of all major computer manufacturers to portability among a wide variety of machines is demonstrated by their involvement in the ANSI standardization efforts. This is very comforting to managers who must deal with a wide variety of systems, many which seem to be outgrown or outdated almost before they are plugged in. If what is written for one machine will run on another, a large part of the anguish of conversion is assuaged. Also important in this respect is C's multiple file model, which allows for a simple division of the portable and machine-dependent portions of a particular application.

Another key issue for management is personnel. Because of its widespread use, C programmers are easy to find (ever try to place an ad for an experienced SPL programmer?) And for re-training existing staff members, there is a wealth of books, magazines, seminars, and courses.

OTHER LANGUAGES SIMPLY CAN'T COMPETE

The other popular compiler languages have well-known advantages, but none offers as potent a combination of features as C. For example, COBOL is a favorite for business applications, but is slow, cumbersome, and useless for systems-level programming. PASCAL, the pride and joy of the academic elite, didn't offer enough systems-level constructs at conception. As a result, mutant offspring were spawned from the original language, each having its own version of systems level extensions. The sum of them render PASCAL

hopelessly non-portable. The fact that FORTRAN was the first high-level language developed almost says enough – statements and data structure techniques have been vastly improved since FORTRAN's inception.

Finally there is SPL, every HP3000 systems programmer's favorite language (until he or she tries C!). Although powerful, SPL has no high-level data structures (such as structures or unions), is extremely machine dependent, has almost zero portability (HP won't even write a SPECTRUM-based compiler for it) and, because some of its operators are operand-dependent, can sometimes be very painful to modify.

SO IS C THE PERFECT LANGUAGE?

Like every language, C has its shortcomings. To concede one point to the anti-C contingent, the language can indeed be abused. The classic example of C abuse is perpetrated by the "hacker", who gets so excited about all of C's features that he chooses a convoluted approach over of the more simple, straightforward one (for example, storing addresses of a series of functions which must be called sequentially inside of an array, and then calling them by indexing into each element of the array, instead of just calling each function by name.) Ultimately, however, this is more of a programmer problem than a language problem. This type of person will generally abuse any language he uses.

No matter how carefully a language is designed for portability, it can be still used in a non-portable way. For example, a program which fills structures with one read from a file may run fine on one machine, then mysteriously blow up on another. "Holes" in structures, required on machines with different byte and word addressing modes, are the culprit. Other hidden machine dependencies are the way chars are packed into ints, signed integers are shifted, and string constants are stored. And the way pointers and function calls are defined and implemented in C, although not a portability concern if used correctly, can cause more than one headache for the wet-behind-the-ears beginner.

ANSI STANDARD TO THE RESCUE

The ANSI standard committee has achieved remarkable results in reducing C's "dangers", without reducing its appeal. Function prototypes were introduced to clear up the insidious "mismatched function call parameters" explosions. All implementation-defined and undefined behavior is well-documented. And, in addition to meticulously describing the syntax and semantics of the language itself, the committee has also provided a clear definition for each library function.

IF I LIKE WHAT I HAVE, WHY CHANGE NOW?

C compilers, readily available on almost all other computers, have finally arrived on the HP 3000, and will be available on the SPECTRUM. The ANSI standard committee on C is close to publishing a standard that has thus far met with very favorable reviews. Even though some productivity loss is inevitable when moving to a new machine, the results are well worth the wait.

Each new program in another language is one more program to be rewritten should a new machine be chose.

C is a powerful, beautiful language, and it is here to stay. So if you're not thinking about using C, maybe you should be.

AUTOMATIC POLLING SYSTEMS FOR MINICOMPUTER/PC NETWORK

Authors: Colin Knight
KSD Systems
6291 Dorman Road, Unit 6
Mississauga, Ontario
Canada L4V 1H2

Alex Tschyrkow
DataSoft International
129 Mount Auburn Street
Cambridge, Massachusetts
USA 02138

Since the introduction of Personal Computers, we have been faced with the challenge of how to utilise the power and performance they offer. This, in turn, has posed the problem of the best way to integrate them into our particular computer environments, and the types of communications required to achieve this integration.

Most managers would agree that significant improvements can still be made in the timely collection and transfer of relevant business data. A manager needs timely information in order to make better decisions concerning marketing, production, staffing, financial planning and other areas.

This paper discusses primarily why automatic polling of personal computers can, in certain circumstances, be an attractive solution, in comparison to other computer solutions, what some of the key features of an automatic PC polling system should be, and several examples of successful polling of PCs.

Several Approaches to PC Data Communications

Alongside the rapid growth of the PC has sprung up a whole new industry of communications specialists offering a myriad of new and not so new products. Some already familiar terms alongside many new ones have taken on an increasing importance in today's computer environment. We are now faced with WATS Lines, Leased Lines, X.25 PAD's, Thick LAN's, Thin LAN's, etc, etc. This, of course, can lead to much confusion on just what is the best method of communications to use.

The confusion is merely illusory as, with all computer solutions, once we break down our objectives into manageable pieces, the communications solutions are usually quite obvious. An important point to note here is that not only must the communications package meet the communications requirements, but its ease of use must match the level of the user who is going to be using it. For example, one of the most common of all the uses of the PC is within Head Office, where users and DP personnel utilise the PC as both a terminal for the HP3000 and a PC to run such programs as LOTUS 123 and DBASE. Obviously you require a product that can handle terminal emulation and PC initiated file transfer, as data will almost certainly be exchanged between the PC and HP3000. It is here we have a potential problem. While DP staff have no difficulty in operating interactive file transfer, there is often a lack of willingness on the part of users to do likewise. How many of you have LOTUS 123 users who have thousands of dollars worth of spread sheets that never get backed up? Could we utilise an automatic method of data collection by the HP3000 to alleviate the problems of PC backup and file exchange? It is a possibility.

We do not have enough time to cover all possible types of communications scenarios so, for the purpose of our discussion on automatic polling systems, we will concentrate on

problems and solutions when using the PC as a remote stand alone work station which requires regular exchange of data between itself and the central computer.

By its nature the PC lends itself to stand-alone usage, and one of the major areas of benefit that can be gained from the PC is the low cost turn-key solution. It is now possible to put PCs into retail outlets, remote warehouses, local or branch offices where they can offer great performance for a low level of investment. The PC is able to carry out stock control, accounting, inventory control, act as an intelligent cash register and so on. Of course this leads us back to the central issue of how to maintain, collect and update the information that is held on these remote machines, in a timely, accurate and cost effective manner.

One approach could be to use the medium of floppy discs. Discs could be sent through the mail to and from head office by regular post. One benefit of this approach is that communications costs are kept to a minimum. Unfortunately, it is likely that we have all suffered the vagaries of the postal service! Despite what they may have us believe, mail does occassionally go astray. This would inevitably lead to lost data and, possibly, unhappy customers if, for example, some orders went missing. Recovery from this situation would almost certainly be difficult. Courier services can be used as a more reliable alternative, but this would significantly increase costs. Perhaps the major drawbacks in this approach are the lack of timely information due to the delays inherent in the mail service, and that such heavy reliance on manual procedures must inevitably lead to a break down.

Another possible approach is to train staff at the remote site on the use of a PC controlled communications product to transfer the required data. This approach has the benefit of requiring nothing more than a modem at each site, and telephone connection only when data transfer is taking place, thus minimizing cost. Unfortunately, even if this process is automated there are a number of problems that the DP manager still has to face.

Firstly, a change in staff at the remote site will require the training of a new person to take over the responsibility for the data communications and may result in delays or, in the worst case, loss of data. Also, the cost of this training is likely to come out of the DP budget.

Secondly, some retail outlets, for example, stay open for twelve or more hours which makes it unlikely that the trained operator will be around at the end of each day. Communications may have to be undertaken during work hours leading to an interruption in the services performed by the PC. In cases where the PC is used as a cash register, such interruption is obviously impossible.

Thirdly, a problem will arise when the number of remote locations surpasses the number of indial ports on the HP3000. To continue the retail example, it is unfortunate that most of the outlets tend to shut at approximately the same time. If you have 40 sites all trying to dial in simultaneously and there are only 5 indial ports on your HP3000, then the problems are obvious. The system will break down because the staff at the remote sites will not wait around all night. You could try to circumvent this problem by arranging specific times for each remote site to call in. Unfortunately, if the PC is required while the outlet is in operation, as with the cash register example, then this will not work.

Lastly, something we consider to be a major problem is one of controlling the exchange of the data. How do we identify what has been successfully sent and received? With data sent to the HP3000, the problem is not so great as we can see what new files have arrived, but what if one of the files has only been partially transmitted? Will we get the same data

plus some more the next day? Will we double up on orders and unneccessarily increase our inventory? Identifying what has been successfully sent to the remote site is very complex - we really have no sure way of telling. This situation is, of course, unsatisfactory.

A third possible approach would be to have all the locations hooked through a Local Area Network and sharing an MS-DOS partition on the HP3000. With the advent of X.25 and some of the new features of HPAdvanceNet this is now possible. The benefits are that communications by file transfer methods are now obsolete and keeping track of the data is made much more simple. Unfortunately the drawbacks are quite severe. A retail chain with 80 outlets would certainly tax any HP3000 system, and it would not be at all cost effective to dedicate an HP3000 system to serve an MS-DOS environment. The communications charges would be high, as an X.25 PAD at each location is quite costly. The HP3000 would also have to be equipped with a number of INP's which, as we all know, are expensive. Finally, a failure of the central machine would lead to the operation of ALL the remote sites coming to a halt. This, of the all the approaches mentioned so far, is certainly the most expensive, and probably the least satisfactory.

A better approach is to look at automatically polling the PCs under the control of the central HP3000. This approach could better be termed the "Centralised approach to Decentralisation".

The major benefit of this approach is that the remote site requires no user to operate the communications system; it is handled by the central computer. This will eliminate the possible problems created by individual responsibility and staff turnover and, at the same time, eliminate training costs. As the HP3000 will determine when to call each remote site, dependent upon the number of ports available to it, and the open/close time of the site, phone line congestion can be eliminated. The process may be automated in such a way that phone lines are used when rates are less expensive since no human intervention is necessary. The central computer utilization is improved by exchanging data during off peak periods. Most importantly we can overcome the problem of tracking the exchanged data as the HP3000 can keep a record of all the remote sites that it has contacted and the data it has transferred.

It would appear then that in situations where a number of remote sites are going to use the PC in a stand-alone system environment, an Automatic polling system is certainly best suited to control the flow of data between the central site and the PC. Another great advantage is that the DP department is, once again, in control of this flow of data.

Let us now look in more detail at the major requirements for an Automatic Polling System.

Some Essential Features of an Automatic Polling Systems for PCs

Our experience has been mainly in the HP3000 environment and, for this reason, we will discuss in this section the features we believe are necessary for an HP3000/PC automatic polling system. However, our conclusions would also apply to other minicomputers and mainframes.

For purpose of discussion, we will refer to the HP3000 as the minicomputer, and PC as an MS-DOS personal computer. General considerations and the essential features of the automatic polling system will be reviewed for the HP3000 and then the PC. With regard to hardware considerations, we assume that the HP3000 does not have an INP (Intelligent Network Processor) and that the HP3000 communication is done either via a modem/phone

line or hardwired. It is our belief that minimizing hardware requirements, such as eliminating the need of INP, makes the polling systems more attractive.

General Considerations

The automatic polling system should be controlled by an HP3000, though software modules must be installed on both the HP3000 and the PCs in order for the system to be in complete harmony. The HP3000 must act as the operations centre, knowing when to get in touch with the PCs, how to get in touch with PCs, what information to transmit, what to do in case of a problem and to report to the DP Manager statistical information. The polling system should be bi-directional as it should be possible to transfer data automatically in both directions. Also, there should be a manual override if certain conditions change. The system should be capable of transferring all types of files and the data should be compressed for improved speed. The system should be able to handle different makes of modem and multiple occurrences of each of these different types. The highest speeds of modem should be used where possible. Finally, error detection is an indispensible part of the system.

Main Features of the HP3000

The HP3000 should be able to do the following:

- Transmit files from the HP3000 to the PC
- Receive files on the HP3000 transmitted by the PC
- Allow definition of transmission parameters in a control data base:

    . Information (phone number or device number) on each remote location
    . Specification of files to be received or transmitted
    . Number of lines available
    . Type of modem
    . Time of day/night to start transmission
    . Number of dialing retries

- To dial (when phone lines are used)
- Compress data
- Detect errors
- Provide statistical information on transmission
- Facilitate use of system.

As one of the major benefits of a polling system is the ability to control and report the movement of data between the PCs and the HP3000 it would seem logical to store such information where it is easily accessible - an IMAGE database. The information held in the database will be used to drive the system, and as the system is running data will be updated and added to it, thus we will refer to it as the "Control" data base.

The polling system on the HP3000 has three specific functions. For purposes of clarity we will assign each process a name. They will be called the "Grandfather", "Father" and "Son". We will discuss below the different role each has to play, which is schematically represented by exhibit 1. We will then review in more detail the essential features for the HP3000 mentioned above.

"Grandfather" process

The governing process of the HP3000 polling system is the "Grandfather" process whose main resposibility is to create a "Father" process for each type of modem that we will be using on the HP3000 (we include direct-connect PCs as a modem type), since communication procedures may be different for each type of modem. The "Grandfather" process will review the "control" database for relevant information and will then allocate different remote locations to each "Father" process.

"Father" process

The task of the "Father" is to process all the remote sites that belong to its particular modem type. The "Father" consults the control database to determine how many lines are available to it, and, as the actual task of contacting each site is carried out by the "Son" process, the "Father" Process must create a "Son" process for each of the available lines. Once the "Son" processes have been created the "Father" can begin to process the locations under its control. A number of issues must be considered: is the location ready to accept communication based on opening/closing hours? Has the location been put into a "wait for communication" state? Is the location actually operational? And so on.

Once the "Father" has identified a location that is ready for communication, it collects the relevant information about the location and passes it to an available "Son". The "Father" must now monitor its "Son" in order to know when the "Son" has completed its appointed tasks and to know when that line is again available. When the "Son" has finished the data transfer, it passes back some information on the communication session with which the "Father" has to update the statistical database.

Once the "Father" has reached the end of its processing, it reports back to the "Grandfather".

It should be noted that the "Father" process can have multiple "Son" processes under its control. The "Father" must continually monitor its "Son" processes for completion of current tasks, and should allocate new locations to available "Son" processes until all locations have been processed or all time limits have been exceeded, if so specified by the "control" database.

"Son" process

The "Son" process is responsible for dialing and establishing contact with the remote PC location and controlling the transfer of files between the computers.

The "Son" is activated by the "Father" and will control its operations according to the data sent to it by the "Father". Its first task is to attempt to open the specified HP3000 port and then to attempt to dial the location. One of four conditions can occur:

1. There is a succesful connection
2. The line is busy
3. A connection is made but no "acknowledgement to proceed" is received from the remote PC
4. Dialing fails

In case 2, the "Son" will attempt to redial until condition 1 above occurs, or until the number of redial attempts exceeds the pre-defined maximum, as specified in the "control" database. In cases 3 and 4, the "Son" will abort the dialing attempt and return to the

"Father" process for further instructions.

Once the "Son" has a successful connection and has received an "acknowledgement to proceed, it sends some parameter information to the remote PC.

The "Son" will then begin to exchange files with the remote location. Information is passed in blocks, with checksum verification. In the event of corrupted data being encountered, the block will be retransmitted until it is successfully transmitted or the number of retry attempts exceeds the maximum allowed (as specified in the "control" database). If this happens, the "Son" will cut the line and redial the location assuming that the maximum number of redials has not been exceeded. The reason for redialing is to attempt to establish a cleaner line with less "noise". When a line is re-established, the "Son" will attempt to pick up file transfer (receiving or sending) from where it left off.
We will now briefly review the essential features for the HP3000 in order for the "Grandfather/Father/Son" process to function.

Transmission of Files from the HP3000 to the PC and Vice Versa

The polling system must be able to co-ordinate and conduct actual file transmissions from the HP3000 to the PC and vice versa.

"Control" Database for Definition of Transmission Parameters

A "control" database must be established on the HP3000 which specifies the transmission parameters. The control data must contain, at minimum, the following types of information:

- Details concerning the remote location; its device number if it is hardwired, and phone number if it is a phone connection.
- Details regarding the files to be received or transmitted for each remote location.
- Details concerning the lines available for transmitting the data.
- Details concerning the modem.
- The time of day/night when the dialing should commence and finish. Also the number of retries when dialing fails or when transmission is lost in midstream. Also the time delay to start redialing.

Dialing System

When transmission is through phone lines and not hardwired, then the HP3000 must be instructed as to how to dial the remote locations, without additional hardware except for the modems.

Data Compression

The software system should use data compression techniques in order to reduce the duration of transmission.

Error Checking

Integrity of data is vital if the polling system is to be useful. A variety of algorithms are available to control accuracy of data transmitted.

Statistical Information on Transmission

If transmission problems occur, it is important for the DP Manager to know what they are in order to take quick action. The polling system should produce a statistical report once the pollling is completed for the day.

Ease of Use

The polling system should be transparent to the PC user so that it is not necessary to know the technical details. This way the HP3000 DP department involvement is reduced to a minimum. Also, the set-up of the polling system should be made as simple as possible and changes should also be easy to make.

Main Features for the PC

Each remote PC must have a communications module that waits to be activated by the HP3000 ("Son" process). It could also be an option for the PC to have a maintenance and configuration program for defining files to be transferred or received.

The method by which the PC module is activated would depend on the main system that is being run on the PC. It could be embedded as the last program in an "End Of Day" routine or called as a menu choice. Alternatively, it could be activated with reference to the PCs clock or perhaps as a "POP UP" when the HP3000 calls.

If data is being transmitted to the PC, it is then also useful for the polling system on PC to have the ability to call another program on the PC.

In addition, it is important for the PC to have the ability to override the automatic polling system if, for example, the phone number is changed for the PC (i.e. when a sales person takes the PC to another location).

Finally, the PC should be able to report how the transmission was conducted and if any problems occurred.

Case Histories - HP3000/PC Automatic Polling

Below we review three recent case histories of HP3000/PC Automatic polling systems which have been successfully implemented. Many other types of application, however, are possible.

Mortgage Pre-qualification System

A subsidiary of one of America's largest financial companies entered into an agreement with a retail chain that has 2000 outlets to provide a mortagage pre-qualification service in a storefront style operation. It was decided that personal computers would be used at the pre-qualification centres, rather than online terminal, with the whole network to be managed by an HP3000. An automatic polling system was installed with the main features being (see exhibit 2):

1. PC is used in stores to provide pre-qualification analysis through a proprietory computer program.

2. Object code updates and parameter tables from the proprietory computer program are automatically sent to the PCs by the HP3000, via the polling system.

3. The HP3000 automatically polls the PCs every night to obtain the daily customer pre-qualification files.

Supermarket Price Update/Sales Report System

One of Europe's largest supermarket chains developed a system where PCs in the stores are connected to a master cash register (that in turn is connected to upto 20 laser bar code cash registers). An automatic polling system was installed on the HP3000 for data exchange with the PCs. The major aspects of the system are (see exhibit 3):

1. PC collects sales data from the master cash register.

2. HP3000 automatically polls every PC for sales/inventory data.

3. HP3000 automatically sends price and product updates to every PC.

4. PC transmits updates to master cash register.

5. HP3000 consolidates storewide sales/inventory data.

Educational Software Cataloguing System for a School District

One of the larger school districts in the United States implemented an automatic polling system for its 450 PCs using an HP3000 and Office Share Network from Hewlett Packard. The HP3000 maintains a library of educational software available for instructor's computer lab use. Each night the HP3000 polling system dials the computer labs and downloads instructor requested programs to the PCs. Additionally, programs which have been resident on the PC longer than 14 days are purged.

Conclusions

Although automatic polling of PCs with a mini computer is a novel concept, it has been successfully implemented by a number of HP3000 sites, who have found it more cost effective than other methods of PC data communications. Automatic polling systems for PCs plays a vital role for those DP departments that wish to retain control of data transfer, thereby minimizing the involvement of PC users in data communication activities. Improved HP3000 utilisation and more rigourous statistical reporting are extra benefits of such a system. It is for this reason that we suggest when organisations are reviewing data collection and transfer systems, that automatic polling of PCs be at least considered as an alternative approach.

EXHIBIT 1 -- "Grandfather/Father/Son Process for Automatic Polling"

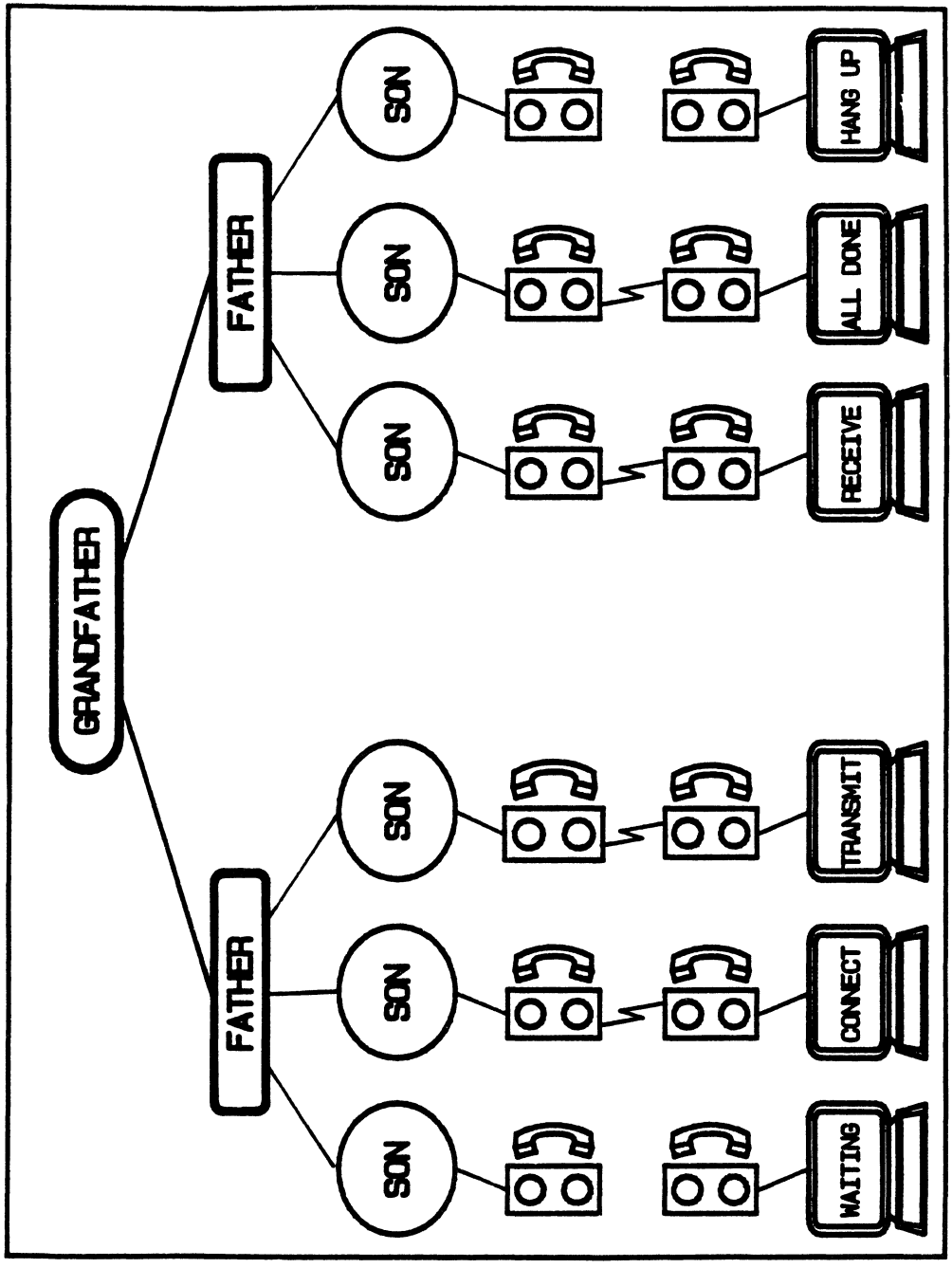
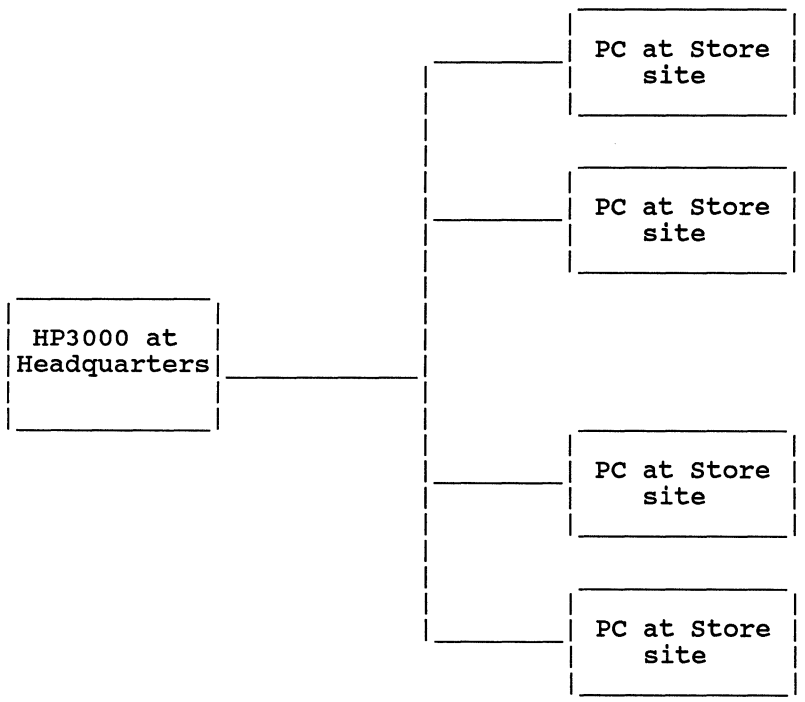


EXHIBIT 2

Automatic Polling for Mortgage Pre-Qualification System



Key features of Mortgage Pre-Qualification System:

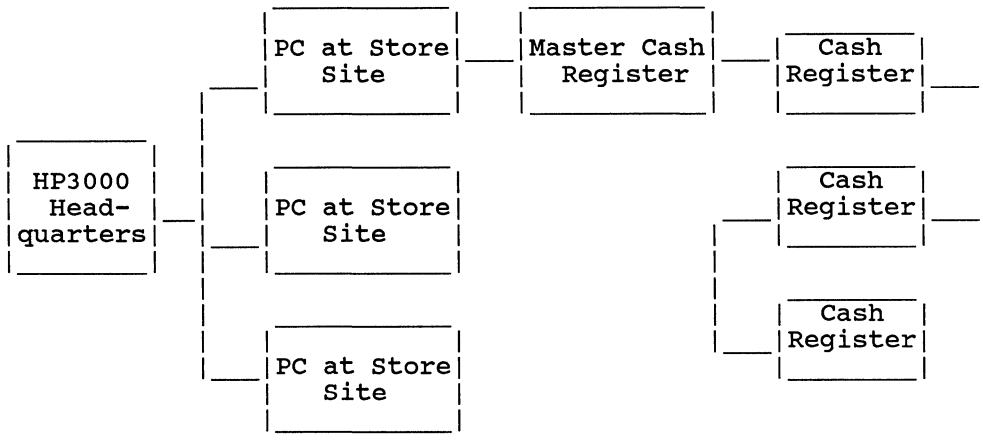
1. PC is used in stores to provide pre-qualification analysis through a proprietary computer program.

2. Object code updates and parameter tables from the proprietory computer program are automatically sent to the PCs by the HP3000, via the polling system.

3. The HP3000 automatically polls the PCs every night to obtain the daily customer pre-qualification files.

EXHIBIT 3

Price Update/Sales Report System for Supermarket Chain



Main Features of Price Update/Sales Report System:

1. PC Collects sales data from the master cash register.
2. HP3000 automatically polls every PC for sales/inventory data.
3. HP3000 automatically sends price and product updates to every PC.
4. PC transmits updates to master cash register.
5. HP3000 consolidates storewide sales/inventory data.

TAKING A SHORT BREAK ...

Michel Kohon
Tymlabs Corporation

"Please, do not interrupt."

Tacked to a door, this notice might refer to a Meeting of the Board, or a poetry reading. But if it appeared on a computer, it would certainly be the HP3000 under MPE.

Although the architecture of the HP 3000 is very well suited for handling interrupts, starting or stopping a process is a painful endeavor for MPE. The :RUN command costs so much that new MPE versions have an auto-allocate procedure. This may be fine for starting a program, but is no help when switching from one program to another.

On the other hand, the BREAK key provides a neat interrupt feature and is cheap in terms of system resources. However, BREAK doesn't provide any way to actually run a second program. It merely gives access to MPE, which most of the time is useless in a business application environment.

I spent a long time at a merchandising company where people live in an interrupt-driven environment. Phones ring, display monitors update commodity prices every 15 seconds, and decisions are made in very short time. If the merchant is looking at his long/short position and a customer calls to get some info on his current shipment, do you think the merchant will wait one minute while his currently-running program shuts down and a new one begins? The answer is NO. He will not use the computer. But, if it could take a few seconds, he would!

We at Tymlabs have been working for some time on the concept of process switching. The purpose of our efforts is to provide the quickest possible way to access a program while running another one. We came early to the conclusion that the BREAK function was the most appropriate means to achieve that goal.

Once having reached that point, we looked for some documentation on BREAK. Will you be surprised if I say that we didn't find any in the whole MPE manual library?

The next step was to investigate the MPE source code itself. By a stroke of pure luck, we managed to find the section which contained the information we needed. Now I would like to share with you what we have learned about BREAK — and what we plan to do with this knowledge.

BREAK: a guided tour

Did you ever wonder what happens when you hit BREAK on your terminal?

I will now describe exactly what does happen, going through the intricacies of the MPE operating system, discovering on the way its sophistication and its weaknesses. Remember that MPE is a multi-process oriented system; be ready to follow BREAK into several sub-processes.

There are two ways to call BREAK — either by pressing the BREAK key on your terminal, or programmatically by calling the intrinsic CAUSEBREAK.

Let's first examine what is happening when you hit the BREAK key on your terminal, assuming that you don't have any OPTION NOBREAK in your UDCs, and that none of the programs you have been using so far called the intrinsic FCONTROL to disable BREAK. (We will deal with these cases later on as the standard situation is complex enough.)

1. The terminal driver detects the BREAK when a terminal generates an extended spacing condition (not a character but rather a hardware condition), causing the ADCC, ATP, etc. to interrupt with a BREAK status.

2. The driver process calls the procedure BREAKJOB. BREAKJOB is an uncallable, privileged procedure which organizes the user process into the BREAK state.

3. BREAKJOB hibernates all the user's sons and sets the user in a BREAK state by turning on a bit in the 1st word of the MPE table called LPDT (Logical Physical Device Table). Then BREAKJOB calls the procedure CAUSESOFTINT, which runs on the Command Interpreter stack. CAUSESOFTINT is a general procedure in charge of all types of pseudo-interrupts.

In order to switch processes, BREAKJOB sets on the user's process pseudo-interrupt bit in the PCB (Process Control Block) using the procedure SET'PSIF. It then awakens the C.I. whose PCB offset is passed as a parameter to the AWAKE procedure. The C.I. starts executing the procedure CAUSESOFTINT, as its PCB is now marked as being in BREAK.

4. After some more validity checks on the status of the calling process, CAUSESOFTINT calls the entry point SYSBREAK located in the actual C.I. code.

5. If OPTION NOBREAK was specified in one of the active UDCs, SYSBREAK calls FCONTROL to disable BREAK and returns to CAUSESOFTINT which re-activates the user's process. This explains

why you don't see anything happening when you hit BREAK when under the influence of an OPTION NOBREAK. It also means that you cannot enable BREAK with FCONTROL when OPTION NOBREAK is in effect.

6. If there is no OPTION NOBREAK, word 32 of the C.I. PXFIXED (a MPE table built in every stack) is set to -1.

7. If there is an ongoing read at the terminal, or if the program is waiting for a read (FREAD,READ,ACCEPT, etc.) the user's program is engaged in a procedure called IOMOVE which interfaces the high level intrinsic with a low level procedure dealing with logical devices.

This low level procedure is ATTACHIO. ATTACHIO controls the logical transfer of data between ANY physical I/O device and the stack. It is one of the most complex procedures in all of MPE. Actually ATTACHIO is the tip of a deep iceberg which reads from, writes to, and locks any physical device (terminal, HPIB, ADCC, ATP, etc.).

A Word to the Wise: If you have to deal with ATTACHIO, make sure the logical device number is valid — unless you want MPE to grant you a System Failure 206!

ATTACHIO is called by IOMOVE to read the logical device attached to the terminal (LDEV 20 for example). When you hit BREAK on your terminal, ATTACHIO detects it as an error in I/O.

On detection of BREAK, IOMOVE unlocks the terminal file control block (ACB) to let the C.I. process run (unless it is the C.I. itself which is running in which case the BREAK request is voided).

8. The uncallable FBREAK procedure is called (on the C.I. stack) in order to set the terminal file control block (ACB) break bit. The ACB is then locked for a short period of time to insure that no conficting access to the file takes place. FBREAK is running on the C.I. stack.

9. Eventually, FBREAK releases the ACB. If there is a read pending on the terminal, IOMOVE tries to lock the $STDIN ACB. But since we are now running the C.I., this attempt impedes the user's process in a low priority queue internal to the file control block vector ("long wait" state).

10. Once FBREAK is completed, the C.I. checks that to make sure the calling process was a session before displaying the expected colon. If it was a job from which BREAK was called with CAUSEBREAK, the output buffer is flushed.

11. At this point, the C.I. is waiting for an acceptable MPE command. Most commands are permissible except RUN and others which imply RUN. This is understandable since RUN would create a brother process rather than a son (don't forget that the C.I. is now active). But wouldn't it be nice to have some kind of mechanism to run a program from within a break?

12. When the C.I detects RESUME, ABORT, BYE, or HELLO, word 32 of the C.I. PXFIXED is set to zero (no longer in BREAK) and the procedure FUNBREAK is called.

13. FUNBREAK locks the terminal file control block (ACB) and sets its ABORTREAD flag to FALSE. This bit will be used by any waiting IOMOVE. An ATTACHIO is fired to tell the terminal driver to clear its BREAK DIT (Device Information Table) so that the current read (if any) begins with data that has already been input and stored in a pending buffer. The ACB is unlocked.

14. If the C.I. detects a RESUME command, it returns to the procedure CAUSESOFTINT. This brings all user's sons out of hibernation and resets the user's PCB to a no break state.

15. The user's process is no longer impeded, and the program resumes. If a read was in progress, IOMOVE can now lock the terminal ACB to complete that read. But before re-initiating the read, IOMOVE displays the familiar (and hard-coded!) "READ pending". (How about a beep or even a double beep instead of this message printed smack in the middle of a formatted screen.)

This is the end of your guided tour ... BREAK-time is over; your program has resumed.

One more thing. When a program uses CAUSEBREAK to break, all the the above steps occur as described, except that the entry point in the break circuit is step 4 rather than step 1. CAUSEBREAK takes care of hibernating the process's sons and awakens the C.I. via a pseudo-interrupt.

When the Option NOBREAK is specified in a user's UDC, the BREAK function is disabled at the time the UDC is executed. The UDCINIT procedure calls the SETSERVICE procedure which sets the LPDT break bit to 1. This indicates that the terminal is already in BREAK (although it is not), effectively voiding any other BREAK.

As you can see, a tour of the BREAK function is almost as complex as a visit to the Hearst castle and I cannot even be sure that my description is perfect since no documentation is available. In fact, I would be very interested in your comments if you happen to know (and have documentation of) what is actually going on during specific phases of BREAK.

The Riviera interface to BREAK

As of this writing, the Tymlabs product which will take advantage of the results of this research is not yet on the market and doesn't have a name. By the time the printed article actually hits your mailbox, the product and its name may be public knowledge. For now, I will use the code name Riviera (that is where many Frenchmen like to take their breaks).

In order to take control when you hit BREAK or you call CAUSEBREAK, Riviera substitutes its own routine for one MPE procedure which is called during the BREAK process. Our research indicated that the safest place to substitute would be at the point where SYSBREAK comes into play. This was by no means the only possibility, as was certainly not the easiest to implement. A long series of experiments led us to this conclusion — and every one of them causing some kind of system failure! SYSBREAK's advantage is that it is an entry point in the system SL, and therefore can be called as a procedure.

To substitute for SYSBREAK, we perform a procedure called "trapping". To do this, the actual SYSBREAK is renamed and our substitute SYSBREAK decides when and how to call it. We do not change MPE code and we let MPE call our uncallable new SYSBREAK, which is located in a special segment of the system SL.

In order to provide access to Riviera, even when OPTION NOBREAK has been set by a UDC, we also trap the procedure FCONTROL with a substitute FCONTROL. Our FCONTROL decides for itself whether to satisfy a break disable request or not.

Riviera also needs to know who is logged on at a specific terminal. To get this information, we trap INIJSMP which is called by the HELLO command. (Actually, we also trap the procedure STARTDEVICE which controls the user, account, group access in order to provide an automatic logon by logical device. You can say "HELLO" or "HELLO password" and be automatically logged on in your correct session. The password is an optional Riviera password. MPE passwords are prompted for as usual if not specified in the Riviera data base. When specified in the data base, they can be encrypted to insure full confidentiality.)

Description of SYSBREAK

Several times during the description of Riviera which follows, I will use the expressions "if Riviera is active," or "if Riviera is not active." You may wonder how a procedure knows if Riviera is or is not active. Since we don't want a procedure to have to make a disk access in order to determine whether Riviera is running, our only choice is to set a flag somewhere in a core-resident table.

For this purpose, we decided to use the Job-ID field of the JMAT (Job Master Table) related to the session/job which is actually running Riviera. This field is not critical for MPE.

When Riviera starts, it stores in this field the string "$DESK" followed by the Riviera version number. Every time we use the phrase "Riviera is active," it means that a procedure has determined that the JMAT contains one entry with the string "$DESK". Since there is no way a session or job canf log on as $DESK.GROUP.ACCOUNT, and since $DESK cannot be displayed during a SHOWJOB, our trick is both safe and invisible.

When Riviera is stopped by the system manager or operator, the original Job-ID is restored. This is not necessary for MPE's purposes, but is useful for any resource accounting or security system which accesses the Job-ID at log on at log off.

If Riviera is not active when BREAK is called, we call the original SYSBREAK. If the user has access to BREAK, he gets the colon. Otherwise he gets the usual nothing!

If Riviera is active when BREAK is called, we check that we aren't in Riviera itself (we don't want anyone to interrupt Riviera). Riviera disables BREAK also (with FCONTROL) but under no circumstances should we allow a break in Riviera.

If someone hits BREAK, we have to locate the group and account where Riviera is running. This is necessary to open the two message files used to communicate between a user and Riviera. (These two files, BREAK and RESUME, are built by Central Riviera when it starts operation.) To determine the user's group and account, we use a procedure which scans the JMAT until it finds the $DESK Job-ID. This procedure then returns the group and account, and the message files are opened.

Once BREAK and RESUME are successfully opened and if the terminal is an HP model, SYSBREAK stores the screen contents in the C.I. stack (at this point MPE has given us 9K words of available stack). Then we send a message to the BREAK file, which has Riviera as its unique reader. If it is ok, we perform the following 7 steps :

1. Call FBREAK to set the terminal ACB in BREAK state.

2. Call SETSERVICE to disable BREAK.

3. Wait for a message from the RESUME file.

   The RESUME file reads all messages in a non-destructive manner. Then it checks to see whether the terminal specified in the read record matches the user's terminal number. If it does, the message is re-read in a destructive manner, and we skip to step 7. If it doesn't, we pause for 1 second and keep reading messages (if any). The pause is

necessary since we are running in a linear queue and other processes must have a chance to collect their messages which may be placed in front of ours in the message file.

4. Restore the screen previously saved (if running on an HP terminal).

5. Call SETSERVICE to enable BREAK.

6. Flush the terminal I/O buffers.

7. Call FUNBREAK to reset the terminal ACB to normal (producing the less-than-desirable "READ pending" message.)

Riviera is now in control of the terminal, or rather has allocated one of its sons to the terminal. (We shall come back to Riviera itself later on.) At some point in time, Riviera's son sends a RESUME message to the user's process. This message specifies whether we want to go into the regular MPE break or to resume.

First case: we want to call the regular break.

Our SYSBREAK calls the original SYSBREAK and we are in break. If we were under an OPTION NOBREAK, the original SYSBREAK will kick us out and disable break. This is why our substitute SYSBREAK re-enables OPTION NOBREAK when we come back from the original SYSBREAK by calling SETSERVICE.

Second case: we want to resume.

We return from our SYSBREAK and we are back in CAUSESOFTINT which will resume our program.

If, during our SYSBREAK processing, we find a non-acceptable situation, we will call the original BREAK.

PC convenience with HP3000 power

In the last year or so, a new type of software product has emerged in the personal computer market. This new product is actually a collection of office tools or 'electronic desk accessories,' such as an electronic notepad, a mini-spreadsheet, a computerized appointment calendar, an automated personal telephone directory, a computer simulation of one or more specialized pocket calculators, and so on. While none of these utilities is particularly impressive by itself, these accessory collections offer one outstanding advantage: all of the functions are instantly available at the touch of a single key, even if the user is running another program.

In essence, a very small 'desk manager' program is loaded into the computer's memory each time it is turned on. Then, the user runs any application -- word processing, spreadsheet, database management, etc.-- as usual. The desk manager program is activated only when the user presses a reserved key. The desk manager then temporarily suspends operation of any application software so that the user can look up a phone number, jot down a note, or perform a calculation, using any number of the available accessories. When finished, the user presses the reserved key a second time to instantly resume operation of the application program, in the exact spot where the program was interrupted.

The basic technical premise of the desk accessory package -- that its functions are always available, no matter what other application is running -- has important potential for the HP 3000 environment. By offering a set of simple, useful office tools which can be accessed without the overhead of stopping one program and starting another, such a product would expand the utility of the system to its end-users. The hundreds of thousands of individuals who have HP 3000 terminals on their desks could use the computer to help record, organize and report the kind of information which has so far remained on paper or in their heads.

What we plan to do with the information we have gained from Project Riviera is to use the BREAK key to provide access to a desk manager - and a full set of desk accessories - for the HP 3000. Although the desk manager will run as a single program, any number of users will be able to access its functions simultaneously. And in addition to the features available in personal computer desk accessory products, our product will provide additional functionality based on the multi-user orientation of the HP 3000. For example, after jotting down a note, an HP 3000 user should be able to "send" copies of the note electronically to a list of other users connected to the same system. Also, when recording a new phone number, the number should be able to be made available to other users in the same department, or included only in the private records of the indiviual who entered it. The appointment calendars for several individuals should be able to be updated and maintained by one secretary.

We will also support the addition of user-written desk accessories. Once programmed by the user, these accessories will appear on the menus and be accessed in exactly the same manner as the standard tools. For example, a user will be able to program a function to provide direct inquiry capabilities into an important corporate database, allowing online retrieval and display without running a separate program. Or a module could be added to look up zip codes based on street addresses, or to display a catalog of the corporate library -- in short, to provide any simple processing function that would be useful to access in desk accessory mode.

I think you will agree that Project Riviera will have some interesting implications for the way we perceive and use the HP 3000.

Michel Kohon is the creator of Mirage, The Image lookalike data base for Pcs.. He is most well known for his Step By step methodology, an approach for completing software projects on schedule. he has lectured extensively to international and local users groups. He created the french HP 3000 users group in 1977, and chaired it during 3 years. Mr. Kohon is currently Director of business development at Tymlabs corporation.

Course Authoring Languages and Course Delivery Systems

by
John P. Korb
Innovative Software Solutions Inc.
10705 Colton St.
Fairfax, VA 22032

As terminal-based instruction, training, and testing becomes more popular, many users are becomming interested in the packages which allow them to create their own courseware and/or testing modules.

Many approaches are available, each with its own advantages and disadvantages. Some courseware packages read editor files and consist of only a delivery program, interpreting "lesson" files at delivery time. Others compile a lesson source file into an intermediate code which is executed by a delivery program. Text formatting capabilites are included in some packages, while other packages will only display material exactly as entered by the module author. An HP terminal or HP terminal look-alike is required for delivery by some packages while others have the further restriction of requiring a block mode terminal. Menu based authoring is used in some packages, with other packages using editor files containing authoring language statements and text.

What are the factors by which to judge the various approaches? Which are easiest to learn? Which have the greatest flexibility? Which are the most extensible? Which have the greatest security? Which require the fewest system resources and have the least detrimental affect on system performance? These and other questions will be addressed in this paper.

Section 1: The Lesson: Data File or Program Code?

How is lesson material stored? Generally, lessons are either data files or program code. When data files are used they may be in one of many different formats. Some are straight ASCII files with commands in them, probably created with EDITOR and read by a lesson delivery program - these are the so-called "interpreted" data files. Others are binary files created by an authoring program or compiler and which contain pre-processed commands and text for use by a lesson delivery program - these are the "semi-compiled" data files. Then there are the lessons which are actually

program code created by a compiler. Each of these three major approaches has its advantages and disadvantages.

Section 1A: Interpreted Data Files

Anyone who knows a few simple commands can construct a lesson using the interpreted data file approach. All one must do is sit down in front of a terminal and using an editor, create a file containing lesson menus, lesson text, and a few commands. Since there is no compilation to perform, no special software is necessary for creating or maintaining lessons, and the lesson files are ready to use immediately. Generally, the "interpreted" packages contain a fairly simple delivery program. The program reads the lesson file and prints the text as it appears - no formatting is performed by the delivery program. Simple commands tell the delivery program what to do - start a new screen, display a menu, ask a question, "goto" a section of text, etc. In fact, some users have used packages such as TDP/3000 to provide simplistic training by using TDP's "USE" files in creative ways.

Just as interpreted files have their advantages (easy to construct, can be created by almost anyone, no compilation, instantly ready for use), they also have their disadvantages. The simplicity of their construction is usually evident in the cost of the interpreted package. Interpreted packages are usually less expensive than semi-compiled or compiled (program code) packages.

One of the reasons most "interpreted" packages have fairly simple commands is the high cost of complicated commands at execution time. By their very nature, the commands in an interpreted lesson must by parsed each time the delivery program reaches them - repeatedly incurring the parsing overhead. The more complicated the command syntax, the more overhead there is. Generally, little if any formatting is provided by the delivery program. Text is usually displayed exactly as it appeared in the file. The reason for this is simple. If any formatting must be done by the delivery program, it must be done EVERY time the delivery program reads the text and executes the commands. Since formatting text is usually a fairly high overhead operation, repeatedly formatting text in an "interpreted" package can use up a significant amount of CPU time and have a negative impact on system performance - especially if multiple users are using the package at once.

Security is another problem. As standard "EDITOR" files are used, security generally revolves around the use of lockwords - assuming that the package allows for lockwords! Should there be any question and answer sections in the lesson, or any testing, users are bound to start trying to read the lesson file and find the answer - and they probably will succeed.

Section 1B: Semi-Compiled Data Files

Semi-compiled lesson files take many forms. Some are nothing more than V/PLUS forms files. Others are proprietary file structures containing binary codes and compressed binary data. Both have one thing in common: they are the result of the processing of the lesson author's input into a form which is coded in some form and generally not directly human-readable.

In the case of the V/PLUS based packages, the lessons are created with the V/PLUS utility program(s) and delivered with either ENTRY.PUB.SYS, or a program similar to it. The power of V/PLUS is available to the lesson author, but so are all of the limitations of using V/PLUS. First, one must learn to use V/PLUS before attempting to author a lesson. For programmers this is not difficult, but for someone with limited computer background this can be a trying experience in itself. Second, a block mode terminal must be used. This limits the types of terminals which may be used to deliver the lessons, or in some cases, the data communications devices/methodologies over which the lessons may be delivered. Also, while V/PLUS compresses blanks out of the screen images, it still transmits and receives a fair amount of "overhead" data to support the block mode diplay method.

As an advantage, V/PLUS is a supported Hewlett-Packard product and should be around for quite some time. Thus, lessons written using V/PLUS and ENTRY should be maintainable in the long term.

Proprietary lesson file structures are often the result of an interpreted package "growing up" into a semi-compiled package. Proprietary lesson file structures often have many advantages over the V/PLUS approach. First, the file structures can be tailored to their intended use - instruction, training, and/or testing. Also, directories or indexes can be incorporated into the files to simplify directly accessing a topic.

Unlike interpreted packages, semi-compiled files can be coded in binary to save space and reduce execution time, blank compression can be used to eliminate unnecessary storage and data communication, and text can be formatted at "compile" time. Typically the lesson data is pre-formatted so that it can be passed directly (or with minor translation) to the MPE intrinsics. In some ways, this is much like the "emulation" mode to be available on the Spectrum series systems.

Also, the file structures can be enhanced by the vendor without impacting on unrelated applications and various security techniques can be used to prevent unauthorized listing or modification of the lesson file (encrypting data, using privileged mode files, etc.). As a bonus, the vendor is usually free to add additional features or customize the package to suit a specific customer. Often times the entire customer base benefits from these enhancements.

As for disadvantages, semi-compiled files are produced by some sort of authoring program or package. This means that the author must wait while the lesson compiles before being able to test the lesson or turn it over to the users. Because proprietary file structures are used, the files may have some characteristics which make maintaining them more difficult. For example, if privileged mode files are used, the files cannot be copied or renamed without using a special program or utility package (note, however, that this makes their security excellent).

Section 1C: Program Code

Another method of storing and delivering lesson material is by using program code. For example, an entire lesson could be written in BASIC, COBOL, FORTRAN, PASCAL, or SPL. By using program code it is possible to reduce the system resources that delivering a lesson requires (no interpretation, no translation from binary codes to intrinsic calls). Because of the object code compatability of the HP 3000 family, lesson (program) files should be executable long into the future.

Unfortunately, many of the features which make an authoring package or program useful would have to be incorporated in each and every "lesson" program and would have to be developed from scratch. Also, a programmer or trainer with programming experience would be necessary to write the lessons.

There is another method of using program code - generate the code by using a special compiler that reads semi-compiled lesson files and generates MPE program files as its output. Common procedures kept in a USL and linked into the compiled code generated from the semi-compiled lesson file then produces a stand-alone program which runs faster than the semi-compiled lesson file as executed by a delivery program (the speed difference is not dramatic, or even noticable by the average user, but does show up as a slight reduction in CPU time used).

Unfortunately this approach has its drawbacks also. Linking lessons together requires the use of process handling capability, the use of which may require giving the lesson authors PH capability. If frequent switching between lessons is common, the approach may cause serious performance problems. Should the lessons become rather large (assuming the compiler is smart enough to automatically segment the code it generates) the large program size will adversely affect a variety of system tables and system performance, especially if users use the lesson(s) as "quick reference" and frequently get into and out of the lesson(s). Thus, much of the performance gain by compiling to more efficient program code may be offset by other performance problems caused by compiling to program code. Also, security can be a problem. Unless certain information (such as the answers to questions) is encripted in the program source code (and oh what fun that is to do!), security can be no better than that of an editor file.

Both approaches become rather expensive. Either an application programmer "writes" an application program that delivers the training - a hard-coded "print" program with perhaps questions and answers that is written in a standard language (BASIC, COBOL, ...) or a package reads a semi-compiled lesson file and creates a program file. If a programmer is required to write a lesson, the lesson becomes very expensive due to the cost of the programmer. If a package "compiles" code, the package is usually very expensive.

Section 1D: Summary

While each approach has it advantages and disadvantages, certain aspects of the different approaches stand out.

Interpreted packages are simple, easy to use, and generally inexpensive. However, they lack the more advanced features,

formatting, functionality, efficiency, and security of semi-compiled packages or compiled (program code) packages.

Semi-compiled packages are more complicated to use, and generally more expensive. Some offer the more advanced features, formatting, functionality, efficiency, and security lacking in the interpreted packages, but some also are more restricted in the types of devices they can be used on.

Program code based packages can be the most efficient packages with the most features, but are generally very expensive, especially if multiple lessons are involved or if multiple lessons must be linked together.

Section 2: Text Formatting: High-level or Low-level?

Different packages have different approaches to the formatting of data. In general, the differences in formatting capabilities come down to only a few major points.

Section 2A: What was that escape sequence?

First, does the package have high-level commands for starting a new screen, clearing to the end of the screen, positioning the cursor to a specific location on the screen, underlining text, and other terminal functions? If so, creating attractive, functional screens is much easier, takes much less time, and takes far less trial-and-error than if the lesson author must memorize a dozen or more escape sequences and flawlessly key them in in.

Second, does the package have the ability to format your text for you? Can you key in your text and have the package move text from line to line to fill between the margins? Can you have the package center text for you? Can the package leave blocks of text exactly as you keyed them in? All of these are important functions when you consider the time necessary to develop a lesson. Time spent on manually moving text from line to line or centering headings can be more productively used to enhance the material contained in the lesson.

Third, is there a way of manually entering escape sequences for handling those special cases not covered by high-level commands? Ideally, a special user-definable character should be used to indicate that an escape charater is to be

inserted at that point. This allows the author to see the escape sequences in the file without having to keep going into and out of display functions. As an example, a package might allow the "~" (tilde) character to be defined as a replacement for the escape character. Thus, if it were necessary to enter the escape sequence <esc>H<esc>J (which homes the cursor and clears the screen), the author would simply enter ~H~J.

Section 2B: Summary

The level of formatting functions available to the author can have a major effect on the productivity of the author. The better the formatting capabilities, the more productive the author and the better the likelihood of an attractive, well-balanced screen display.

Section 3: The Terminal: The Minimum Common Denominator?

What kinds of terminals are required to utilize the authoring and delivery packages? Do you need a graphics terminal? Is a block mode terminal required, or will a character mode terminal suffice? Will the software run over your data communications equipment? Must you write lessons to run on the terminals with the minimum capabilities, or can the package accomodate several different functional terminal groups within one lesson?

Section 3A: Data Communications

In the HP shop terminals generally fall into two catagories: character mode devices and character/block mode devices. As their names imply, character mode devices communicate on a character-by-charater basis, block mode devices transfer data in blocks. What is more significant is that while virtually every HP terminal can run in character mode, certain models cannot operate in block mode. Thus, packages which must run in block mode or which use V/PLUS cannot be run on some models of terminals. While the number of models is small, the number comprises a significant percentage of the available terminals at some sites, and may dictate the type of authoring and delivery package used.

Also note that not all block mode operations are supported when terminals are connected to a host system over an X.25 link. Often the users who are connected to a system over an X.25 link are those who are located at a remote location and who are least likely to have other training methods or

instructional materials at hand. Thus, they are generally the least able to survive on their own, and most likely to be hurt if a block mode only solution is chosen.

Section 3B: Terminal Features

What about terminal features? Some users will have very basic terminals with little memory and few advanced features. Others will have terminals with much memory and line drawing and math character sets, and still others will have graphics terminals or personal computers. Authoring lessons for only a specific terminal model can severely limit the usefulness of the lessons as the lessons may be difficult to read or understand on other terminal models. However, attempting to write lessons for the most elementary terminal model can negatively affect the usefulness of the result. Often times it is more effective to write lessons for each individual terminal model. Some packages allow an even better approach.

With some authoring packages the author can specify something like: "display this screen on model "x" terminals, this screen on a model "y" graphics terminal, and this screen on a model "z" terminal". The author can then write common non-terminal-specific portions of the lesson and where necessary use separate sections to handle each of the different types of terminals. In such packages the delivery part of the package identifies the model of the terminal and then selects and displays the appropriate screens for the terminal at hand.

Section 3C: Summary

Formatting options are an important part of any authoring package. Through the use of high-level formatting options the productivity of the lesson author and the effectiveness of the resulting lessons can be greatly increased.

Section 4: Creating The Lesson: Menu Based or File Based?

Generally lesson authoring packages are of the menu-driven type or the source file driven type. Menu-driven packages use a series of menus from which the author defines the type of screen, the options in effect for the screen, and the content of the screen. Source file driven packages use an editor file containing the text and commands necessary to describe the lesson. If the delivery package is of the semi-compiled data file type or the compiled program code

type the editor file is passed through an authoring program which "compiles" the text and commands into an output file, be it a data file or a program file. Opinions vary as to which package is better, or even appropriate for a specific lesson, often leading to lively debates. Below are some of the advantages and disadvantages of both approaches.

Section 4A: Menu-driven Authoring

In a menu-driven approach the author decides the type of screen to be created: a menu screen, a text screen, a question and answer screen, etc. The authoring package then displays a basic screen of the requested type and asks a series of questions. For example, if a menu screen is being created, the authoring package might ask for the menu heading and its justification (left, right, centered). Then it might ask for the keywords to be accepted and the text to accompany them. For each acceptable keyword the authoring package would then ask for the action to be taken. The menu-driven approach is favored by many authors, especially those with little or no computer background, as there are very few commands to learn and little need to be conscious of any underlying logic.

There are three major criticisms of the menu-driven approach. First, the author is forced to proceed through screens again and again. Often the author makes only a few keystrokes on a screen before moving on to another option screen and perhaps another option screen and then finally reaching a screen for entering the text for a screen. The first few times this is not much of a bother, but as the author becomes more experienced the delays associated with repainting screens and repeatedly answering the same questions becomes rather tiring.

Second, the choices of menu packages tend to be more restrictive as to what the author can and cannot do. The author must select from one of the provided options. If the options do not include the function the author wishes to perform, the author has no recourse but to redesign the menu or screen to use one of the options available.

Third, menu packages often restrict the author as to what formatting may be performed. As an example, menu-driven authoring packages often prevent the author from overlaying new text on top of an existing display, or clearing a portion of a screen and overlaying new text in the cleared area (some menu-driven packages do allow the use of a previous screen as a basis for a new screen, with the

delivery program repainting the entire screen, even if only one or two characters have been changed).

Menu-driven packages do have their advantages though. First, they are easy to learn to use - the first-time author is led through the process of developing the screens. Second, the lessons tend to have a standardized "look and feel". Third, the screens are usually displayed to the author in a "what you see is what you get" form. Finally, errors in command syntax are usually caught by the authoring package at entry time.

Section 4B: Source File Driven Authoring

In source file driven authoring, the author uses one of the many editors available for the HP 3000 to enter the text and commands comprising the lesson. Generally, the lesson text is entered as plain text - no surrounding quotes or other indicators - and commands for the authoring package are placed on a separate lines with some special character or sequence of characters at the beginning of the line to indicate to the authoring package that the line is a command line and not text. Those familiar with TDP/3000 and other text processors for the HP 3000 may recognize a certain familiarity with this approach - in fact, certain authoring packages recognize some of the TDP/3000 formatting commands.

Just like the programming language compilers, these packages often have and "include file" feature which allows material from other text files to be included into the lesson at "compile" time. This is particularly handy if there are lesson segments which are common to several lessons. The common material can be placed in a separate file and "included" into each of the lessons using the material.

If much of the text of a lesson already exists as portions of various document files, using a source file driven authoring package can be quite an advantage. An editor can be used to extract the desired text and place it in separate files. The separate files can then be merged and any left over formatting commands removed or converted to the authoring package's formatting commands. Then any menus and authoring commands can be added. This approach can save considerable time over re-keying text into a menu-driven authoring package.

Also, there tend to be more formatting options and commands available in a source file driven authoring package, and there are fewer restrictions in how they may be used. In

some packages it is possible to create very complex structures which would be very difficult or impossible to create with a menu-drive authoring package, or to utilize special capabilities of certain terminal models.

With the added capabilities and features of source file driven authoring come some drawbacks. First, since formatting is usually performed during the "compilation" of the lesson, the author usually can only guess what the final screen can look like - the source file does not reflect what the user will see on the screen. Second, error checking is performed during the "compilation" of the lesson, and thus, the author is not immediately aware of errors made in constructing the lesson. Third, some authors are overwhelmed at the number and complexity of commands available - a menu-driven authoring package with its limited options can be better in such cases.

Section 4C: Summary

Both menu-driven and source file driven authoring packages have their advantages and disadvantages. Menu-driven authoring packages can be easier to learn but generally offer fewer options, features, and capabilities than source file driven authoring packages. Menu-driven packages generally respond to command errors at the time of entry, while source file driven packages catch errors at "compilation" time. Menu-driven packages often force the author to sit through multiple screens of forms to define a single lesson screen; source file driven packages often allow a screen to be defined with a single line command. Source file driven authoring systems are not as restricted as to how options can be combined in new ways.

Section 5: Special Features

Special features can have a major impact on the usefulness of an authoring package. Here are some of the important factors to consider:

* Does the authoring package allow the author to set time outs on screen input?
* Can the input echo be turned off and on (as in simulating password prompts)?
* Can the user return to the previous menu at any time?
* Can the user backspace a screen?
* Can the user enter the lesson at any of multiple lesson entry points (if permitted by the lesson author)?

* Can the delivery system execute MPE commands (if permitted by the lesson author)?
* Can programs be run from the delivery package (if permitted by the lesson author)?
* Can the author specify a special initialization routine to be executed anytime anyone enters the lesson?
* Can the lesson be restricted to only those users having certain capabilities (ie. only users with SM [System Manager] capability can use the System Manager lesson)?
* Can the local site "extend" the capabilities of the authoring and delivery package by writing their own code (written in a language such as COBOL, FORTRAN, or SPL) to be called from a lesson at run time?
* Can the author perform mathematical calculations? Can numeric input be used in the calculations?

Each of these capabilities can be of use to the lesson author. The better the capabilities, the more flexability the author has in creating a first-class lesson.

Section 6: Conclusions

The selection of an authoring package depends on the personnel who will be using the system, their level of experience, the type of equipment in use, the data communication methods used, the types of lessons to be written, the use to which the lessons will be put, and the performance considerations which the package must live up to. In order to make an informed decision, the purchaser of an authoring package must consider:

* The Ease of Learning the Authoring Language
* The Relative Speed of the Authoring Process
* The Cost of the Authoring System
* The Performance of Delivery Package (response time and CPU time)
* The Commands, Functions, and Features Available
* The Devices the Delivery Package Must Support (character mode terminals, or block mode terminals for example)
* The Level of Formatting Commands Available (underlining, centering, filling between margins, etc.)
* The Security of the Lesson File (the security of any questions, answers, or other sensitive material contained in the lesson)
* The Ease of Linking Multiple Lessons Together
* The Ability to "Include" Text from Other Files
* The Productivity of the Lesson Author

* The Range of Applications of the Authoring/Delivery Package
* The Ability to Write Your Own Extentions and Call Them from a Lesson
* The Ability to Issue MPE Commands, Run Programs, and Perform Calculations

Store-and-Forward Network Two Years Later - Lessons Learned

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

Two years ago the paper "Store-And-Forward Data Transmission in a Multi-System Network" was presented at the Washington D.C. Interex Conference. The paper described a system entering production which provided a standardized interface to the application programmer and the ability to "ride out" line outages by transmitting data across as many lines as are "up", storing the data when the next line in the path is "down", then forwarding the data when the line comes back "up".

While the backbone of the network described has remained the same, the network has evolved considerably in the last two years. Timing considerations, DS/3000 bugs, IPC file problems, queueing problems, overloaded DS lines, and frequent X.25 line outages have provided a dynamic development, test, and production environment (all concurrently). As a result, many new features and capabilities have been added to the network to increase its capabilities, reliability, and throughput.

This paper presents the lessons learned and features added as the network expanded beyond twenty systems, crossed a continent and an ocean, and matured to include a mixture of leased lines and X.25 links.

Recap:

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

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

"design and maintenance nightmare" when large networks of fifty or more systems with tens or hundreds of paths are involved.

The network referenced in this paper is a store-and-forward network with a set of user-callable procedures providing a standardized interface to the application programmer to be used for transfering data from any point in a network to any other point in the same network.

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

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

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

What is the Network?

The Network is a collection of programs, procedures, files, databases, and job streams which when properly configured provide a data transportation system with the capability of

transporting data from any point in the Network to any other point in the Network.

What is its Purpose?

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

What are its Features?

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

program determined interval (1 to 255 seconds) for data, then timing out and returning a "no data available" error to the application.

For more information, please see the paper "Store-and-Forward Data Transmission in a Multi-System Network" in the 1985 Washington D.C. INTEREX conference proceedings.

The Basic Components of the Network

The Network Software consists of three major modules:

1) The user interface (the Network Procedures).
2) The packet switching and transmission code.
3) The maintenance and utility code.

The user interface provides eight procedures which the applications programmer uses to write data to or read data from the network.

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

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

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

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

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

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

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

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

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

Environmental Requirements

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

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

The Network is accessed through eight Network Procedures. These procedures are standardized and are the same on all HP 3000 systems in the Network. Using these procedures, one opens the Network, then reads data from or writes data to the Network, and when done, closes the Network. The Network Procedures set up and maintain the extra data segments(s), buffers, data base, and files necessary for interaction with the Network.

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

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

The Iceberg

The major part of the network is a set of three programs which form the transportation services for the network and the files accessed by the three programs. All files accessed by these three programs are IPC files (with the exception of a message catalog file). Below is a brief description of the operation of the transport system and the three key programs as they were initially.

The first program is called the Traffic Control Supervisor Program (TCSP). The TCSP is the brains of the network. It reads a Network Configuration Data Base (NCDB) and determines which processor it is running on, which DS lines are available to communicate with the outside world, and the pathing to be used to transmit data from its system to each of the other possible destination systems. A batch job (which logs on as THE,DATACOMM.NETWORK) is used to execute the TCSP.

When the TCSP begins executing it makes sure that it was properly shut down the last time it was run, that the network's maintenance job is run if it is necessary to clean up any "messy" situation, and then builds the extra data segments used to keep pathing and statistical information handy. As the pathing information is loaded into extra data segments the TCSP checks for configuration conflicts. Should a conflict or other configuration error be found the TCSP calls an immediate network holiday and tells the console operator to get help.

When the TCSP is satisfied that all is as it should be, it creates son processes to handle the data switching and data transmission functions. The TCSP sets up special command

and information buffers in a shared extra data segment for communicating with each of its son processes.

The first process created is the Traffic Control Program (TCP). The TCP acts as a data switch. The TCP simply reads data from its input queue (the Traffic Control Program Input Queue [TCPIQ]), examines the data's destination address, and passes the data off for transmission to a remote or for local pickup by the user-callable NETREAD procedure.

After creating the TCP the TCSP creates a Network Message Transmitter for each logically adjacent system with which the network must communicate (note that two or more "logically adjacent" systems may be accessed over a single X.25 "connection"). Thus, if a particular node of the network is configured to communicate directly with five other systems, the TCSP will create five NMT's. The NMT's each read from their own Network Message Transmitter Input Queue (NMTIQ) and write to the TCPIQ on their remote system. By having separate NMT's for each remote system being directly communicated with, the problem of a single line hang stopping all data transmission is avoided.

When all the son processes have been created, the TCSP one by one activates each of NMT's, telling each one the DS line name to use as well as other information pertaining to the line. After all the NMT's have been activated and "started", the TCP is activated and given a command to start processing.

At this point the network transport system can be said to be "up". The TCSP now begins checking for commands from its own input queue, the Traffic Conntrol Supervisor Program Input Queue (TCSPIQ) and executes or ignores the commands as necessary.

When it is time for the network to shut down, the TCSP receives a "shut down" command through the TCSPIQ and immediately begins sending commands to the son processes (TCP and the NMT's) asking them to shut down. When all sons have responded and shut down (or the commands have timed out) the TCSP terminates and the network is down.

In order to detect data losses and insure the integrity of the network, packet numbers are assigned to each block of data transmitted by the network. The TCP is in charge of assigning packet numbers to data packets created by applications running on the local system. The packet numbers are obtained from an IPC file. When the IPC file

becomes empty, the TCP detects the end-of-file and issues a command to the TCSP to "add more packet numbers".

As an aid in debugging, the TCSP was given the ability to recognize certain command packets as containing MPE commands. This allowed us to set JCW's withing the TCSP's batch job, or send ourselves messages to make sure things were working.

With this basic setup on each of six systems, the SLEEPER utility was used to start up the network and shut it down at various times during the day with few if any problems.

However, as additional systems were added to the network, several problems came up.

The Problems

Problem 1

The first problem encountered was that of shutting down one of the systems in the network while the other systems remained up. As an example, when a new release of MPE was installed it was necessary to have the network up on all systems except the system undergoing the MPE installation. With all the systems at only a couple of sites the procedure was to shut down the network on all systems connected to the system undergoing the software installation, then :DOWN the DS lines, then start the network back up again. When the network expanded to a dozen systems scattered across the country, this involved long distance calls between the computer operators and a lot of long-distance coordination.

The solution was to have the network software communicate "line commands". When the network software on system "A" shuts down, it now sends a "please shut down your line connecting to me" command to each of the systems to which it is directly connected. Likewise, when the network is brought up it sends a "please establish a connection with me" command to each of the systems to which it is directly connected. This approach has eliminated virtually all of the long-distance phone calls and has made the operation of the network much more automatic.

Problem 2

The second problem we came across was that of preventing the network from trying to communicate with new systems which had been configured into the network but which were not available for production use. As initially implimented, the

network would immediately try to establish a connection to every adjacent system configured into the Network Configuration Data Base (NCDB). This meant that as soon as the NCDB was updated to include a new site the network would try to communicate with the new site - often before the new site was ready. Delaying the configuration of the NCDB caused problems in that pre-production testing could not be accomplished.

The solution was to add a special DS line status - "Out Of Service" - which the network software would recognize as a special case. When checking the NCDB's configuration, the TCSP would detect the "Out Of Service" line and create a NMT for the line, but not tell the NMT to establish a connection. Thus, the NMT for the "Out Of Service" line was dormant, ready to be used, but not connected to the remote. This had the additional advantage that the TCSP was receptive to a "please establish a connection with me" command for the "Out Of Service" line, and if such a command were received, the TCSP would command the NMT to open its line and communications would be established.

To facilitate all the commands which were now flying back and forth between the network processes on the different systems, the NMT's were recoded to bypass the standard data queues for certain commands. This meant that the NMT now had two IPC files open on its remote system - one for data and one for commands.

Problem 3

Enter problem number three - IPC file corruption on busy systems when multiple IPC files are accessed remotely. During testing we had the new NMT running for weeks with no problems. When we felt confident with the new NMT, we installed it on a production system - and the problems began. Three days after the installation of the new NMT the application people came by wanting to know where their data had gone. Then the network's own internal check of packet numbers declared that there was a "packet sequence error". The new NMT was removed and the old NMT re-installed.

Testing of the new NMT on a different account began. In effect there were now two networks running at the same time on the same systems - one for production, and one to test out the new NMT. After a couple of days there was a data loss on the test network but not on the production network, implicating the new NMT. The HP Response Center was called for help. As a result we coded up a very simple program which opened two remote IPC files and alternately wrote data

to them. This program ended up having more serious problems than the NMT had had. It hung, it corrupted the files on the remote system, and it lost data.

Unfortunately it ran just fine on the HP systems at the response center. Our systems were cold started. No improvement. MPE was completely re-installed on our systems. No improvement. Since HP could not duplicate the problem, there was no action taken by HP. Our solution: we recoded the new NMT to never have two remote files open at a time. It currently FCLOSEs the data queue, FOPENS the command queue, sends the command, FCLOSES the command queue, then FOPENs the data queue. This requires the systems to perform much overhead, but it avoids the problem.

Problem 4

Another problem related to the IPC files was caused by the operators. IPC files can be very severely damaged by a system failure if the file label and file structure on disc are not current when the system fails. "BAD VARIABLE BLOCK STRUCTURE" is one of the more common problems, and is often irrecoverable. To keep the IPC files current on disc an FCONTROL 6 is issued. In theory, if an FCONTROL 6 is issued after every operation, the file should be kept current on disc and the IPC files should survive system failures. Due to sensitivity to the possibility of a system failure between the time a network process reads a record from one queue and writes it to another, the network is designed to take advantage of an option of IPC files called "non-destructive read". Non-destructive read allows some overlap time between the time:

1) the Network Procedures read a packet, and the time they delete the packet from Network storage

2) the Traffic Control Program (TCP) reads a packet and the time it writes the packet to an Application Output Queue (AOQ) or Network Message Transmitter Input Queue (NMTIQ)

3) the Network Message Transmitter (NMT) reads a packet and the time it completes writing it across its DS line to the TCPIQ on the remote system.

Ideally, using non-destructive reads provids a safeguard against data loss due to program or system failure. How? Here is an example of the basics of the operation of the NMT (Network Message Transmitter).

The NMT performs a non-destructive read from its input queue, the NMTIQ. Since a non-destructive read has been performed, the NMT has simply received a copy of the first record to be read from the NMTIQ. The NMT then writes the data to the TCPIQ on the remote system and follows the write with an FCONTROL 6 which forces the file buffers, control blocks, file label information, etc. to all be updated on disc. The data is now (we hope) safe on disc on the remote system. Now the NMT performs a destructive (ordinary) read against the NMTIQ to delete the data it just moved across the DS line. The destructive read is followed by an FCONTROL 6 to make sure the file buffers, control blocks, file label information, etc. are updated for the NMTIQ. At no time during this operation is the data only in memory - there is always a copy of the data on disc on one system or the other, and for a short time interval, on both at the same time.

Enter disc caching. The operators were forgetting to enable "BLOCKONWRITE". As a result even the FCONTROL 6's were being cached. This meant that all this wonderful code to protect the IPC files and the data was going for naught.

The solution was to have the network job (THE,DATACOMM.NETWORK) issue a variety of operator commands to ensure that the operating environment was as it should be.

Problem 5

Then came controller caching - and one of the current mysteries. Every so often (months apart) large amounts of data are lost when there is a DS related system failure. At first we blamed DS, but then a strange coincidence was noted - the only systems loosing any data were those with controller caching enabled! As yet we have no clues as to what the cause of the problem is, but by analyzing the contents of the network's extra data segments from the memory dumps it appears that the data that the network processes think have been confirmed to be on disc somehow never made it.

Problem 6

Another problem started when the load on one of the central systems increased - multiple network jobs would be running at once but no data would be transmitted and any user attempting to access the network would immediately hang.

This was very distressing because code was included in the TCSP to avoid the execution of two copies of the network at

the same time. The TCSP issues a :TELL command to DATACOMM.NETWORK (the user.account the network job stream logs on under) and checks the command interpreter error message it receives in return. Depending on the value received the TCSP either continues to execute (the sender is the only "target") or sends a nasty message to the console ("only one TCSP can execute at a time!") and aborts. Somehow this code was not working! But how!

Then another mystery appeared. If you entered a :SHOWJOB command you would see multiple copies of the network job in the :SHOWJOB listing. If, however, you tried to send :TELL commands to each of the jobs you quickly found out that all but one did not exist! Next OPT was used to check on the job streams. Guess what! Only the last job streamed existed as far as OPT was concerned. Conclusion: The network jobs were shutting down, but never completely going away.

What was preventing them from terminating? System log file dumps indicated that the jobs would progress to a certain point in the shutdown of the network and then nothing additonal would happen. It appeared that when it came time for a network process to FCLOSE the TCPIQ something would prevent the FCLOSE from completing. To get a better idea of what was happening inside MPE, we asked the operators to halt the system and take a memory dump whenever the multiple network problem came up. A couple of memory dumps later we had found the problem - a timing problem within MPE relating to IPC files. A letter and a copy of a memory dump were sent to HP, and about a month later a patch was installed which cured the problem.

Problem 7

The next problem is a bug/feature of DS. As long as you open and close DS lines from the colon prompt, DS works as documented. If, however, you open DS lines from a son process, beware! We have a situation where the TCSP creates the multiple NMT's which each open DS lines to different systems and then transmit data over those lines.

If you open a DS line from the colon prompt and there is a line problem, you can simply close the line and re-open it to reconnect. If you open a DS line from a son process and there is a line problem, you CANNOT simply close the line and re-open it! Instead, the son process must terminate and a new son process must be created to re-open the line.

This caused a lot of additional code to be generated for the network, and results in considerable additional system

overhead (inter-process communication "my line is dead and I'm terminating", process termination, process creation, process activation, inter-process communication "you are...you transmit to...please connect to the remote").

Problem 8

Some of our data communication links are leased lines, and others are over an X.25 network. The X.25 service we use is a non-commercial (ie. U.S. Government) service. The X.25 network we use is slow, drops connections at least once a day, keeps losing its packet size configuration information, and is generally a great source of frustration.

When commands are sent from process to process within the network job stream, the sender will wait up to two minutes for a response before it declares the receiver a "non-responding software module". As we gained experience with leased lines we tuned the NMT to try to transmit as much data as possible before checking for an inter-process command. With X.25 lines we have had to modify the NMT to transmit only one-fifth as much data between command checks (for X.25 links). Even now we still have frequent command time-outs.

Problem 9

As briefly mentioned in problem 9, the X.25 nodes keep forgetting the packet sizes used by our HP systems. Our X.25 packet sizes are 512 bytes. This size was chosen after a couple of months of experimentation. Initially we used the default X.25 packet size which yielded very poor transmission rates. A user typically took minutes to log on, and a DSCOPY from coast-to-coast of a moderate size file was often impossible - the connection would drop after several hours and the user would have to restart the DSCOPY... again and again. We tried using an X.25 packet size of 1024 bytes, but DSCOPY did not work, so the size was cut back to 512 (while the network does not use DSCOPY, other users do).

One day we started hearing that several of the systems were having performance problems. We were about to investigate when the operator at a site in the midwest called. She was quite irate. It seems that the network was continuously logging onto and off of her system from several remote sites with such rapidity that she could not enter any commands on the console. We logged onto her system (over a leased line) and found that we had a five digit session number, even though the system had been started up that morning! Then we noticed that it was only the remote network sessions coming

into her system over the X.25 net that were logging on and off, so we logged off of the leased line and established a connection to her system over the X.25 net. After the WELCOME message printed and a colon prompt was received, a DSCOPY was attempted. Almost immediately there was a DS error message and the connection was broken.

The NMT writes a log record to a circular file whenever it detects a line problem, giving the NMT module detecting the error, the file system error number, and the date and time. This file was examined on a couple of the systems and it was found that there had been a major outage of the X.25 network just before everyone arrived at work in the morning. Armed with this information the X.25 liason contacted the provider of the X.25 service and asked what was going on. Apparently a couple of the X.25 nodes had power failed and had lost the packet size information relating to our HP systems.

With multiple network remotes trying to connect and reconnect, the signon and signoff messages were coming in faster than the console's printer could handle them. On the remote systems the inter-process communication, process terminations and process creations were occuring so frequently that they were saturating the systems (series 70's). Setting the network's configuration information to indicate the X.25 lines were "Out Of Service" brought things under control until the X.25 provider corrected the X.25 node configurations the following night. Score one more for X.25.

Problem 10

Years ago we defined the names of the procedures which the users would call to access the network. We noted that MPE modules for the various commands were prefixed by "CX'" as in "CX'RUN", "CX'SPL", "CX'LISTF", etc. We were very careful to avoid using names similar to those HP uses for its modules. We named our procedures NETOPEN, NETCLOSE, NETREAD, NETWRITE, NETEXPLAIN, NETCONTROL, NETINFO, and NETSTATUS. Then one day NS software arrived. Guess what! There was a :NETCONTROL command. Sure enough, there was a CX'NETCONTROL procedure in the system SL.

Then one of the systems people tried entering the :NETCONTROL command. He was greeted by one of our network's error messages stating that the status array was corrupt. Well, at least our Privileged Mode procedure had done its checking and had avoided a system failure, but what was the conflict?

A copy of the system SL was made and CX'NETCONTROL was decompiled. Guess what! CX'NETCONTROL calls LOADPROC to try to load a procedure named... you guessed it, NETCONTROL! Oh boy! At present we are some months away from installing NS on any of the systems, and so have a conversion period during which all the applications programs referencing NETCONTROL can be converted to use the new name NETALTOPEN. The name NETALTOPEN was selected because most of what NETCONTROL does could be accomplished by calling NETCLOSE and then NETOPEN with differing parameters. Also, both NETCONTROL and NETALTOPEN have names the same length. Thus, should any production programs slip through the conversion process, a program to change the program's external list to replace NETCONTROL with NETALTOPEN can be used.

Enhancements to the Network

In addition to the enhancements made to respond to problems the following enhancements have been made:

* Most console messages generated by the network have been removed to eliminate operator confusion.
* The command and control functions of the network have been expanded to allow any system in the network to issue commands to any other system in the network. Some of the commands implimented are:
  * Shut down. The network on any system can be shut down from any other system by command.
  * Shut line. Any NMT anywhere in the network can be directed to close its line to its remote.
  * Open line. Any NMT anywhere in the network can be directed to open its line to its remote.
  * Who are you. Any TCSP anywhere in the network can be directed to return to the sender the identifier of the system responding, the software version of the network running, and the current date and time.
  * Start system. Any NMT anywhere in the network can be directed to start up the network on an adjacent remote.
  * Issue MPE command. Any TCSP anywhere in the network can be directed to issue an MPE command. This means that jobs can be streamed in Hawaii by a command issued in Washington D.C. without anyone having to DS across to HAWAII to issue the commands. Common uses of this enhancement are the purging of files, building of new accounts, issuing of operator commands, and streaming of jobs.

* The network now reports packet sequence errors not only to the system console, but sends a statistical packet back to the primary development machine. This enhancement was made because we are rarely notified of system failures. As far as the network is concerned it is often the case that the only indication that there has been a system failure is that the packet statistics are out of sync. Since the operators rarely reported the network's packet sequence error messages, we often only found out about potential problems weeks after they occured.
* The TCP and NMT's now keep statistics as to how many packets they have moved and how many words of data have been moved. From this we can see which lines are most heavily used.
* The NMT and TCSP now cooperate to automatically establish a new NMT and connection to a remote after a line problem.

Planned Enhancements to the Network

* Rather than rely on SLEEPER for scheduling, a network scheduler will be developed to coordinate scheduling across systems and across time zones.
* Transmission statistics will be sent back to the primary development system for reduction, analysis, and reporting.
* Transmission statistics will be sampled at specific intervals along with measurement of the queue lengths. The samples will be sent back to the primary development system for reduction, analysis, and reporting. One of the proposed reports is a report showing when the throughput of the transmission lines becomes less than the data volume flowing into the network and queueing results. It is expected that the reports generated from this information will assist the hardware staff in ordering appropriately sized data communication lines.
* The inter-process communications between the TCSP, NMT, and TCP are presently serial in nature. The inter-process communication code will be changed to support paralled command processing.

Lessons Learned

With approximately twenty-five systems installed and one or two additional series 70 systems being installed each month, the network is still slightly less than half its final size.

As the network has matured it has become more flexable in terms of use and abuse. Generally, once the software is brought up on a system it runs without maintenance or problem reports for months. The automatic recovery code of the TCSP handles most system failures and operator induced errors (except RELOADs from the wrong set of tapes, an all too common occurance). Most irrecoverable errors (data loss) have been directly attributed to MPE problems, most of which have been patched by HP.

If while researching a possible approach to a problem you find that the documentation on the software package (say DS) is vague about specific points, it is probably because there isn't anyone who knows what the package will do under those circumstances. In such cases all you can do is experiment and try to develop your own theories. If you are lucky your experiments are an interesting experience, otherwise... welcome to the twilight zone.

When you run across a problem that you think may be caused by a bug in MPE or DS (or whatever software), provide the most documentation on the problem to get the speediest and most complete solution (send memory dumps, dump of system log files, documented listings of the code being executed, etc.).

Learn how to use IDAT, DECOMP, and your trusty (but sometimes inaccurate) System Tables Reference Manual to figure out what MPE was trying to do when: (1) the system hung; (2) the system failed; (3) DS locked up; or (4) all your processes are waiting for something that's never going to happen.

Most importantly...
Have a sense of humor - remember that mere mortals like yourself wrote MPE (and DS, and ...), and they were under a deadline too!

Diogenes -- Searcher For an Honest Data Base

Jim Kramer
Hewlett-Packard Company
9606 Aero Drive
San Diego, CA 92123

Introduction

Diogenes is a new Image and TurboImage data base integrity checker. It has the same purpose as the classic program DBCheck: to detect structural defects in an Image or TurboImage data base, such as broken chains, invalid keys, etc. Such defects can be caused by disc failures, system interrupts, or software defects (Image or MPE).

Diogenes can do exactly the same checking as DBCheck. However it has a new checksum check on chain pointers which eliminates, in most cases, the need for chain following. This can reduce the time needed to check a detail data set by a factor of ten or better.

Diogenes is also restartable; checking can be aborted and then restarted at a later time to continue from where it left off. This allows data bases to be checked even when there is no single period of time long enough to allow a complete check.

My hope is that the speed and restartability of Diogenes will encourage people to do regular checking of their data bases, rather than waiting until programs quit working.

Diogenes is being distributed to Customer Escalation Centers, and may eventually be available in the TELESUP account. For now, if you want it, please ask your SE or CE to get it for you from his Escalation Center.

This paper is a description of Diogenes and the checking it does. Since Diogenes is doing, for the most part, the same checking as DBCheck, this is also a description of that program. So many of us have run DBCheck without quite knowing what it does!

This paper contains: a feature comparison of Diogenes and DBCheck; descriptions of the checking they do; a performance comparison; mention of a new data base repair program named Scalpel; and some concluding remarks.

Comparing Features of Diogenes and DBCheck

Since DBCheck is a classic program in the HP3000 community, and since Diogenes is intended as its successor, I will introduce Diogenes by comparing it to DBCheck.

Diogenes is faster. Even when exactly the same checks are done, Diogenes appears to be faster by 20% or so. When chain following is eliminated, which is feasible because of the checksum check done on chain pointers (described below), Diogenes can be faster by a factor of ten or better.

Diogenes is more flexible. DBCheck allows checking of an entire data base or a single set of a data base. Diogenes allows checking of any selection of sets from multiple data bases. Moreover Diogenes defines three checking phases for detail data sets, and allows selection of any set of phases for any detail.

Diogenes is restartable. If a Diogenes run is aborted, it can be restarted and checking will continue from where it left off without loss of work. DBCheck does not have this capability.

Diogenes allows concurrent read access to the data base. When Diogenes is checking, users can access the data base being checked by specifying DBOPEN mode 6, a read-only mode. DBCheck allows no concurrent access.

Diogenes and DBCheck both prompt the user to supply the required checking specifications (data base and set names, etc.). DBCheck's dialog is short and simple. Diogenes' dialog is longer and less obvious, but Diogenes will supply help on request at any prompt.

Diogenes allows customization of text. Diogenes obtains its text, including prompts, help text and error messages, from a separate message catalog. This means that the text can be easily modified, even translated to another language. The text used by DBCheck is embedded in the program, and cannot be changed except by reprogramming.

Diogenes will, on request, generate a separate line printer listing. DBCheck only writes to $STDLIST.

When checking an entire data base, Diogenes does the fast serial checking of all sets first, postponing the slow detail set chain checking to last. DBCheck does all checks of a set before going on to the next set.

Diogenes will, at the user's option, generate a batch job that can be streamed to do the data base checking. This frees the terminal for other uses. DBCheck can be run in batch but will not generate its own job stream.

Diogenes' error messages are intended to be more easily understood than those of DBCheck. The messages are also generally briefer, because Diogenes only lists the part of the entry that actually contains the error. DBCheck often lists an entire entry when only a part of it is bad.

Checking Phases

Both DBCheck and Diogenes have the following four checking phases:

For master sets:
a serial pass.

For detail sets:
a serial pass.
a delete chain check.
one or more path checks.

When DBCheck checks a set, it does all phases for that set (one for a master, three for a detail). For detail sets the order of checking is the order listed above. If checking of the entire data base has been specified (by entering /A in response to the data set prompt), the sets are checked in the order in which they appear in the schema.

When Diogenes checks a master set it does essentially the same serial pass as DBCheck. However for a detail Diogenes allows any combination of phases. Moreover, for the path checking, any set of paths can be specified.

When Diogenes is asked to check an entire data base (by use of the /A response), the user is given a choice of whether path checking is to be done. If it is, the path checking is postponed until all the serial and delete chain checking has been done on all sets. The advantage to this is that the very slow path checking does not postpone the relatively fast (and fairly thorough) serial checking.

The Checksum Check

The only major checking innovation in Diogenes is the checksum check, which eliminates most of the need for chain following in detail sets. This checksum check is done during the serial pass for both master and detail sets. For a master the check is done on the synonym chains; for a detail it is done on the path chains.

In order to understand the checksum check, consider a sample synonym chain in a master data set:

Entry Number	Synonym Chain Count	Backward Pointer	Forward Pointer
2	0	5	9
5	0	0	2
9	0	2	0
11	4	9	5

The primary entry is 11, as indicated by the non-zero chain count. The chain of secondaries in forward order is: 5, 2, 9.

The important thing to notice is that every secondary is pointed to twice, once by a backward pointer and once by a forward pointer, and the primary is not pointed to at all. Thus the entries in the backward pointer and forward pointer columns are identical. This means that we can check the integrity of the chains by doing separate checksums on the forward pointer and backward pointer columns, and comparing the results for equality. Chain integrity can be checked by a simple calculation performed during the serial pass through the data set, which both DBCheck and Diogenes do anyway.

This important and simple (once you've seen it) idea was given to me by Dave Morel, an HP Systems Engineer from New Zealand.

The same idea, with slight modifications, can be applied to the chain pointers in a detail set. The problem here is that there is no forward pointer pointing to the first entry of a chain, and no backward pointer pointing to the last entry of a chain. Here is an example detail set chain:

Entry Number	Backward Pointer	Forward Pointer
6	8	13
8	0	6
13	6	0

Entry 8 is the beginning of chain; there is no forward pointer that points to it (except in the chain head of the associated master set). Entry 13 is the end of chain, and there is no backward pointer pointing to it. The fix is fairly simple. When we encounter a beginning of chain entry (identified by a zero backward pointer) we manufacture a forward pointer to point to it to use in the checksum. So, in the example, when we encounter entry number 8, we manufacture a forward pointer of 8 to use in the checksum. A similar fix is used for end of chain entries, and when we encounter entry 13 we manufacture a backward pointer of 13. Thus for checksumming purposes, our chain looks like this:

Entry Number	Backward Pointer	Forward Pointer
6	8	13
8	0	6
		8 (manufactured)
13	6	0
	13 (manufactured)	

Now we have both a backward and forward pointer pointing to every entry, and the checksum idea works.

Currently Diogenes computes the checksum by doing a bit-by-bit exclusive or on the 32 bit pointers. This may change.

The checksum check is a very good check on chain integrity; there is very little chance (about one in four billion) that arbitrary modifications to chain pointers would leave the backward and forward checksums equal.

However it should be mentioned that the checksum technique is not as thorough as path checking (described below) for the following reasons:

The checksum technique does not check the key values on the chain.

The pointers from the chain heads to the chains are not checked.

It does not disclose the location of the bad pointers. Of course one can elect to do a path check if there is a checksum error.

Still I think the checksum technique is good enough that path checking can be omitted in most cases. There are tremendous speed advantages (see the performance comparison below).

Master Set Serial Pass Check

Diogenes and DBCheck do the same checking on a master data set, except that Diogenes also does the checksum check of synonym chains as discussed above.

The data set is read serially from first entry to last. Both free entries and data entries are read. Whether an entry is considered to be a data entry or a free entry is determined by the bit map which resides at the beginning of each block of entries.

A free entry is checked to be sure that it is null, i.e. that all its words are binary zeroes.

A data entry has the following structure:

Synonym Chain Info -- Chain Head Info -- Data (Including Key)

The Synonym Chain Info comprises a synonym chain count, a backward pointer, and a forward pointer. The following checks are done on the Synonym Chain Info:

The synonym count must be less than or equal to the number of data entries in the set.

Each backward and forward pointer must either be a valid pointer which points to an entry within the data set, or be zero. A zero pointer indicates that the current entry is either a primary (and there are no secondaries) or that it is a secondary pointed to by a primary.

There must be consistency between the chain count and the pointers:

If the chain count is zero, the data entry is a secondary; either the forward and backward pointers must be unequal, or both must be zero.

If the chain count is one, the data entry is a primary and there are no secondaries. Both pointers must be zero.

If the chain count is two, the data entry is a primary and there is one secondary. The pointers must be equal and non-zero.

If the chain count is greater than two, the data entry is a primary and there are multiple secondaries. The pointers must be unequal and non-zero.

The Chain Head Info comprises zero to sixteen chain heads. Each chain head has a chain count, and two pointers which point into the detail set for the path. One is the backward (or end of chain) pointer, the other is the forward (or beginning of chain) pointer.

Chain heads are checked for consistency between the pointers and the chain count. The pointers themselves are not checked to be sure that they are valid pointers into the detail set (a desirable enhancement). Consistency between the pointers and chain count means:

If the chain count is zero, the forward and backward pointers must be zero.

If the chain count is one, the forward and backward pointers must be equal and non-zero.

If the chain count is greater than one, the forward and backward pointers must be unequal and non-zero.

If the master set is an automatic, at least one chain head must have a non-zero count (otherwise the master entry should have been deleted).

One of the most important checks Diogenes and DBCheck do is to use the key from a data entry in an attempt to fetch the entry by a mode 7 (hashed or calculated) DBGET. The entry found must be the same entry as the one from which the key was taken. This check does two things: it checks the validity of the key and the integrity of the forward paths through the synonym chains. Neither Diogenes nor DBCheck check the integrity of the backward paths through the synonym chains. Nor does Image, I believe, check their integrity while doing a mode 7 DBGET.

When the serial pass is completed, Diogenes and DBCheck have three counts of the number of data entries in the set, and these must all match. The counts are: the number of data entries encountered during the serial pass (the number of block map bits that are set), the sum of the synonym chain counts, and the value recorded in the user label (obtained from DBINFO).

Detail Set Serial Pass Check

Diogenes and DBCheck do the same checking during the serial pass through a detail data set, except that Diogenes also does the checksum check of path chains as discussed above.

During the serial pass through the detail set, every entry is read from first to last, up to the capacity of the data set. Both data entries and free entries are read.

First consider the case of an entry that is free (available) according to the bit map. If it is found in front of the high water mark it should be linked into the delete chain. In this case the first two words should be zero or be a valid pointer to another entry, and this is checked. Moreover, the entry should be null (binary zeroes) following the pointer.

If the entry follows the high water mark, it is checked to be sure it is null.

If the entry is a data entry it must precede the high water mark. The chain pointers are used in the checksum check as described earlier. Moreover, validity checks are done on the chain pointers:

Each pointer must be zero or be a valid pointer which points to an entry that precedes the high water mark.

The pointers must not be equal unless both are zero.

When the serial pass is completed, the number of data entries found (according to the block bit maps) must equal the count of data entries in the user label (obtained from DBINFO).

Detail Set Delete Chain Check

The delete chain is a chain of entries which have been deleted by DBDELETE and not yet re-used to provide an entry for DBPUT. The pointer to the first entry of the chain is in the user label. The first entry of the chain has a pointer (located in the first two words of the entry) to the second entry, the second has a pointer to the third, and so on. The last entry of the chain has a zero pointer.

Diogenes and DBCheck follow this chain, starting at the user label. To guard against loops in the chain, delete chain checking stops if the number of entries read exceeds the capacity of the set.

The entries are checked to be sure they are null following the pointer. No special check is done on the pointer except to use it to find the next entry.

When the end of chain has been found, the count of entries found is compared to the number expected. This is calculated from information from the root file and data set user label which is provided by DBINFO. The calculation is: (Capacity - High Water Mark - Data Entries).

Detail Set Path Checking

A detail data set has from zero to sixteen paths. For each path there is a separate set of chains; a data entry in a detail data set will be linked into as many chains as there are paths.

Path checking for Diogenes and DBCheck is the process of following all all the chains of a path. When DBCheck is asked to check a detail set it automatically checks all paths.

With Diogenes there is more flexibility. If Diogenes is asked to check an entire data base (/A response to data set prompt), then the user has the option of checking all paths in all sets, or checking no paths at all. The path checking, if selected, is done last, after all serial and delete chain checks are done on all sets.

If Diogenes is asked to check a set rather than the entire data base, then the user may select any set of paths to be checked.

Checking a path means following every chain of the path. The chain heads are found by means of a serial pass through the associated master set. Starting at the chain head the chain is followed forward, entry by entry, until the end of chain is found. The following checks are done:

The backward pointer of a chain entry must point to the previously read entry.

The search item value in the entry must match the value found in the chain head.

When the end of chain is found, the backward pointer in the chain head is checked to be sure it points to the last entry of the chain. Also the count of entries found is checked against the count found in the chain head.

When all chains have been checked, the count of all entries found on all chains is checked against the user label count of all data entries in the set, provided by DBINFO.

Performance Comparison

I have not attempted to do a comprehensive performance characterization of Diogenes. The following results are presented just to show how much time may be saved by using Diogenes instead of DBCheck.

The following table shows checking times for Diogenes and DBCheck for two data sets of a data base:

				Diogenes Serial		Diogenes Paths		Diogenes Total		DBCheck Total	
Set	Paths	Capacity	Entries	CPU	Wall	CPU	Wall	CPU	Wall	CPU	Wall
A	3	75000	58413	37	110	934	3283	971	3393	1372	3900
B	4	101004	101004	52	85	939	2587	991	2672	1703	3480

The times are in seconds. The delete chain check times were zeroes -- apparently the chains were very short.

Diogenes total times are less than those of DBCheck. However, because of the checksum check, Diogenes does a very good job of integrity checking during the serial pass. The serial check times for Diogenes are only about 2% to 3% of the total times for DBCheck.

Scalpel, a Data Base Repair Tool

Data Bases are getting larger, and they are doing it faster than discs are getting faster. As a result, repairing data bases by rebuilding all the linkages is becoming less feasible. I expect the ability to manually repair data base damage to become more valuable.

Scalpel, a program written by Maya Kinariwala of the Fullerton Customer Escalation Center, is a valuable aid to the process of data base repair. It is a data base editor that makes it easy to fetch, display and modify any entry in a data base. It is expected to be available in the TELESUP account.

Scalpel and Diogenes work well together. For instance, Diogenes does not list entire entries when a defect is found, but Scalpel makes it easy to view them if desired. And when repair work is done with Scalpel, the speed and flexibility of Diogenes make it easy to check the work.

Conclusion

Diogenes can do the same checking as DBCheck, but do it faster. The checksum check makes Diogenes' serial check of a detail data set almost as good as DBCheck's complete check, with a tremendous speed advantage.

Because of its speed, flexibility and restartability, Diogenes makes it possible to do regularly scheduled data base checking in situations where it may not have been possible before.

HP TO DEC NETWORKING

GERARD F. LAMEIRO
HEWLETT-PACKARD
3404 E. HARMONY ROAD
FORT COLLINS, CO 80525

DEC, VAX, VMS, DECnet, MicroVAX, MicroVMS, DEQNA, and DELUA are trademarks of Digital Equipment Corporation.

ABSTRACT

HP and DEC share many common customers. Many of them desire to interconnect computers from both vendors. This session will discuss the HP strategy for connecting HP computers to DEC computers. The discussion will focus on capabilities for the HP 1000, HP 3000, HP 9000, and HP Vectra PCs communicating with DEC VAX computers. It will include information on communications based on industry and de facto standards such as IEEE 802.3 and ARPA networking services and information on the HP AdvanceNet product for the DEC VAX computer. It will also include a review of recent relevant product announcements.

INTRODUCTION

Hewlett-Packard is committed to networking based upon industry standards, such as the International Standards Organization Open Systems Interconnect (OSI) reference model. This means that HP AdvanceNet products provide a high degree of lasting value for both networking hardware and software; it also means that standards-based multi-vendor communications are facilitated.

This paper specifically looks at Hewlett-Packard networking products that permit communications between HP 1000, HP 3000, HP 9000, and HP Vectra PC computers and DEC VAX computers. It addresses both asynchronous solutions over RS-232 lines and local area networking solutions that include products which provide ARPA and Berkeley networking services and which rely upon IEEE 802.3 In addition, HP's Shared Resource Manager product and connectivity to DECnet are discussed. Figure 1 below outlines the connectivity alternatives we discuss in this paper.

For each of the alternatives in Figure 1, we will look at the computers and operating systems connected, as well as some of the capabilities and advantages of the alternative.

ASYNCHRONOUS SOLUTIONS

Asynchronous connectivity provides the simplest and lowest cost alternative to connecting both HP personal computers and workstations (as well as HP terminals) to DEC hosts. In the case of general asynchronous solutions (see Figure 2), speeds of up to 9600 baud are supported between most HP workstations and most DEC computers. For these needs, HP offers terminal emulator products.

The products are best applied to situations where low traffic, occasional use, and smaller file transfers predominate. Also, only ASCII files are supported. The simplicity and flexibility of these products along with the fact that they can be used to provide both local and wide area communications are the reason for widespread use.

Another of the asynchronous communications solutions are based on UNIX. HP 9000s running under HP-UX can connect to both DEC VAX/BSD UNIX and to DEC/VAXs operating under ULTRIX. UNIX offers cu to call another UNIX system, uucp for UNIX to UNIX copies, uux for UNIX to UNIX executions, as well as send mail when electronic mail routing is needed. See Figure 3.

This is probably the most widely available UNIX only solution available for communications. As with general asynchronous communications, it supports both local and wide area connectivity and is best applied to low traffic applications.

LOCAL AREA NETWORK SOLUTIONS

In addition to asynchronous solutions, there are four major local area network solutions available today. These include HP's Shared Resource Manager, Berkeley networking services, ARPA networking services, and HP AdvanceNet networking services. Let's look at each in turn.

HP SHARED RESOURCE MANAGER

The Shared Resource Manager (SRM) system is a file and printer/plotter server for HP 9000 workstations and for the HP Vectra PC. It provides the capabilities to share resources among workstations in a local cluster. It supports HP 9000 Series 200, Series 300, Series 500 in addition to HP 9845s and HP 9835s. Depending on the workstation, BASIC, Pascal, and HP-UX are supported. Figure 4 illustrates an SRM environment that permits DEC VAX connectivity.

In Figure 4, an HP 9000 Series 300 or 500 running HP-UX acts as a gateway to permit communications from HP workstations on SRM to DEC VAX computers running either BSD UNIX, ULTRIX, or VMS. Note that the HP 9000 gateway and the DEC hosts are connected to the same LAN.

This approach is probably the best high speed alternative for HP BASIC and Pascal workstations. For applications requiring frequent use of the gateway or for user convenience, it is even possible to automate the SRM - LAN connection.

BERKELEY NETWORKING SERVICES

Another approach that can be taken to connect HP 9000 Series 300, Series 500, and Series 800 computers to DEC VAX/BSD UNIX and DEC VAX/ULTRIX is to utilize Berkeley networking services. On the HP 9000 Series 300 side, these services are bundled into HP 50952B -- a product that includes ARPA networking services, Berkeley networking services, as well as HP AdvanceNet networking services. For the Series 500, an HP referenced third party product is available from the Wollongong Group, WIN/H9000. Berkeley services are also available from HP for the Series 800. See Figure 5.

UNIX commands that are available include rlogin for remote login, rcp for remote copy, remsh for remote shell, rexec for remote command execution, and sockets for interprocess communications.

The 10M bit/sec Ethernet link provides a high speed communications path that can handle high volume traffic effectively and efficiently. In addition, the product provides a high degree of transparency. For UNIX to UNIX, this is probably the best alternative.

ARPA NETWORKING SERVICES

For HP to DEC networking, a user can also utilize ARPA networking services. These services include file transfer protocol (FTP), TELNET (a virtual terminal protocol), and SMTP (an electronic mail protocol). Figure 6 illustrates this type of connectivity.

To obtain ARPA services on the HP 9000, HP provides the HP 50952B product mentioned earlier that includes HP AdvanceNet Network Services (NS) and Berkeley networking services all bundled together for the Series 300. For the Series 800, both ARPA and Berkeley services come bundled together. The

Wollongong WIN/H9000 product mentioned earlier also includes ARPA services for the Series 500.

HP Vectra PCs also can run ARPA services using another third party product FUSION sold by NRC.

To use ARPA services, the HP 9000s require an HP LAN link product; the HP Vectra PC requires the use of a 3Com interface board.

On the DEC side of the connection, DEC sells WIN/VX (also from Wollongong). This product runs on DEC controller hardware or hardware that is sold by Micom-Interlan. Other third party packages are also available for DEC VAX computers.

One of the major advantages of ARPA networking services is its widespread use. It is a de facto standard that permits connectivity to ARPANET and DDN networks. It even supports VMS through third party software.

HP ADVANCENET NETWORKING SERVICES

The final approach to networking HP to DEC computers is using HP AdvanceNet Networking Services (NS). HP AdvanceNet products are available on HP 9000 Series 300, 500, 800 computers as well as on HP 1000s, and HP 3000s. HP has also developed HP Network Services for the DEC VAX (HP NS/VAX).

HP NS/VAX integrates DEC VAX/VMS computers directly into HP AdvanceNet networks. It provides network file transfer (NFT) between HP computers and DEC computers. Specifically, HP 1000 computers running RTE, HP 3000s running under MPE, as well as HP 9000 Series 300, 500, and 800 under HP-UX can communicate with DEC VAXs running under VMS and DEC MicroVAXs running under MicroVMS. See Figure 7.

Also, HP NS/VAX requires the appropriate LAN hardware on the HP systems in question. For example, an HP 3000 would require a LAN/3000 link product. Similarly, the DEC computer would require appropriate hardware such as a DEQNA controller on the MicroVAX or a DELUA controller on a VAX computer. Third party hardware is also available for the DEC VAXs.

Some of the key features of the HP NS/VAX product include its ability to integrate the DEC VAX computers into HP AdvanceNet networks as mentioned above as well as to do transparent and reliable file transfers. In addition, it can co-exist with DECnet systems on the same Ethernet/IEEE 802.3 cable. HP

NS/VAX also adheres to VAX/VMS Digital Command Language (DCL) grammar making it easier to learn and use for customers already familiar with DCL.

The user interface also includes additions to DCL which include dscopy for copies between HP AdvanceNet and DEC VAX computers; npowerup for configuring a node on the network or displaying its configuration; nusers to limit or monitor the remote users of the node; nlinkloop for testing the connectivity to one other node on the LAN, on nodes producing IEEE 802.2 packets; nreadstat for reading controller registers on Micom-Interlan controllers; and nclearstat for clearing those same registers.

Another feature of the HP NS/VAX product is that it is possible to transfer files to and from remote DECnet nodes. See Figure 8.

Also, HP NS/VAX provides record level file manipulation eliminating the need for file formatting utilities. It provides programmer access to NS allowing for unassisted process execution.

Finally, its user interface is tailored to each native system making it a friendly product to use.

SUMMARY

As can be seen from the above discussion, there are a wide variety of approaches that can be taken to permit communications between HP and DEC computers. Currently, there are general asynchronous connections using emulator products as well as UNIX to UNIX asynchronous solutions. In addition, HP 9000s on HP's Shared Resource Manager can be used to act as a gateway to DEC VAX computers. Also, ARPA and Berkeley networking services products are available. Finally, HP AdvanceNet Networking Services can also be placed directly on DEC VAX computers to provide network file transfers between DEC VAX/VMS computers and HP 1000s, HP 3000s, and HP 9000 Series 300, 500, and 800 computers.

Multivendor Networking HP to DEC

1. Asynchronous Solutions

General

Unix to Unix

2. Local Area Network Solutions

Shared Resource Manager

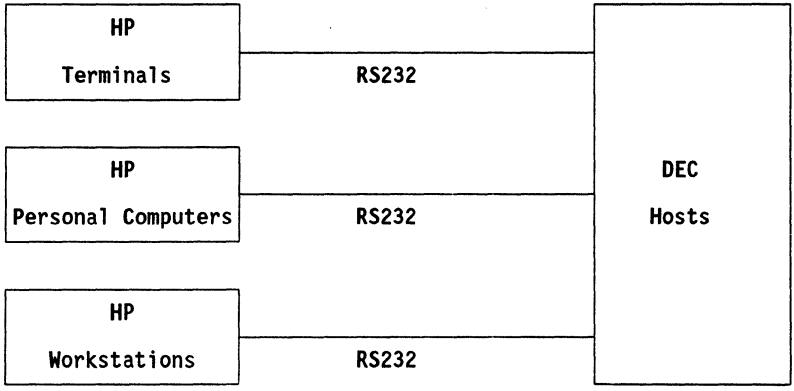
Berkeley Networking

ARPA Networking

HP Advancenet

Figure 1.

GENERAL ASYNCHRONOUS SOLUTIONS



Terminals and terminal emulator products:

HP personality for HP hosts (supports block mode)

ANSI personality for DEC hosts (supports VT100 escape sequences)

Login, run applications

File transfer capabilities (upload and download)

X-modem and Kermit support error checking

Flexible configuration options

Direct connect or modem support

Speeds up to 9600 baud

Advantages:

Supports most HP workstations and terminals

Supports all DEC hosts

Both local area and wide area connectivity

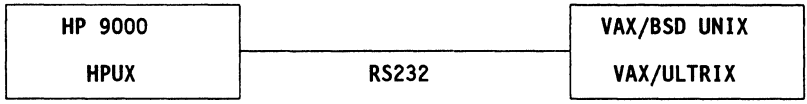
Low traffic, occasional use, small files

Lowest cost

ASCII files only

Figure 2.

UNIX ASYNCHRONOUS SOLUTIONS



UNIX commands:

cu: call another unix system

uucp: unix to unix copy

uux: unix to unix execution

sendmail: electronic mail routing

Advantages:

Most available UNIX only solution

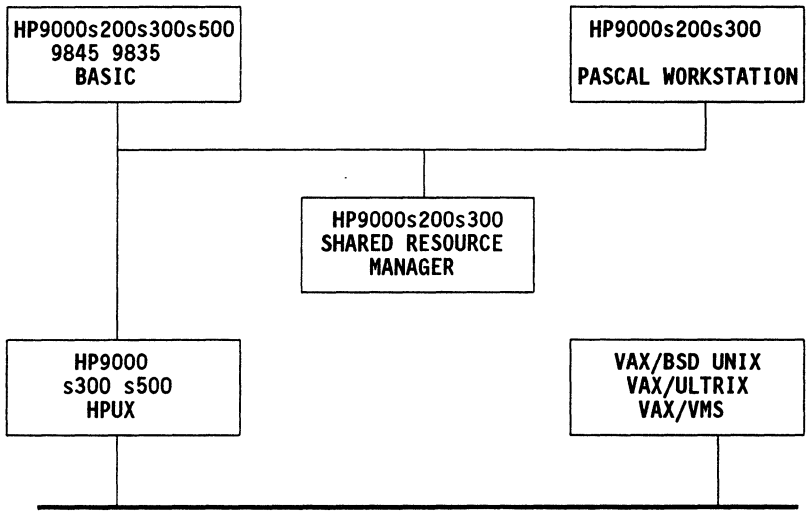
Both wide area and local area support

U-modem and Kermit available for reliable file transfers

Low cost, low traffic

Figure 3.

WORKSTATION LAN SOLUTIONS



Advantages:

Best high speed alternative for HP workstations.

Daemon programs can automate transfers through HPUX gateway.

ASCII files only

Figure 4.

BERKELEY NETWORKING SERVICES

UNIX commands:

```
rlogin:   remote login
rcp:      remote copy
remsh:    remote shell

rexec:    remote command execution
sockets:  interprocess communications
```

Advantages:

Best Unix only solution

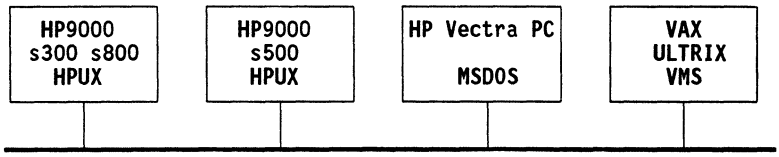
Fast 10M bit/second Ethernet link

High traffic

Transparency

Figure 5.

ARPA SERVICES



Configurations:

SYSTEM	SOFTWARE	HARDWARE
HP9000s300	ARPA Services/300	LAN/300
HP9000s800	ARPA Services/800	LAN/9000
HP9000s500	Wollongong: WIN/HP9000	LAN/9000
Vectra PC	NRC: Fusion	3-Com
VAX	Wollongong: WIN/VX	DEC, Micom
	NRC: Fusion	DEC
	Excelan	Excelan

ARPA commands:

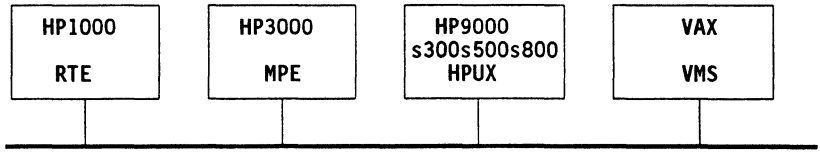
FTP: File Transfer Protocol
Telnet: Virtual Terminal Protocol
SMTP: Electronic mail routing

Advantages:

De facto standard in engineering world
Connectivity to ARPAnet, DDN networks
Supports VMS (through 3rd party software)

Figure 6.

HP ADVANCENET



SYSTEM	SOFTWARE	HARDWARE
HP1000	NS/1000	LAN/1000
HP3000	NS/3000	LAN/3000
HP9000	NS/9000	LAN/9000
VAX	NS/VAX + DECnet	DEUNA/DELUA/DEQNA
	NS/VAX	Micom-Interlan

Network Services

NFT: Network File Transfer

Advantages

Supports HP1000 and HP3000 key systems

Supported now on VAX/VMS with HP's NS/VAX

Most flexibility for file and record translations

Transparent and Interchange modes

User interface taylored to each native system

HP quality and worldwide support

Figure 7.

DECNET LEVERAGE

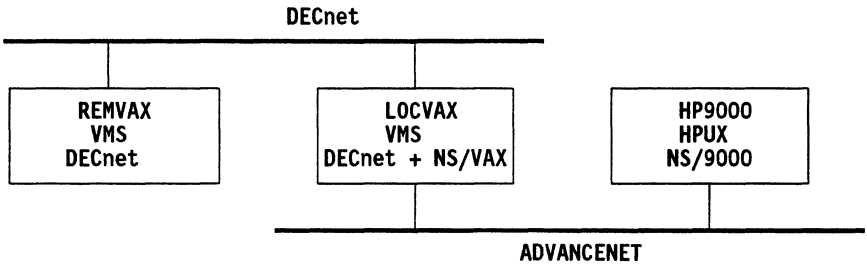


Figure 8.

RELATIONAL DATABASE: HOW DO YOU KNOW YOU NEED ONE?

Orland Larson
Hewlett-Packard
Cupertino, California

ABSTRACT

The field of relational technology is clearly misunderstood by a large number of people. One major obstacle to acceptance of the relational model is the unfamiliar terminology in which relational concepts are expressed. In addition, there are a number of misconceptions that have grown up in the past few years concerning relational systems. The purpose of this paper is to define those terms, correct some of those misconceptions and to help you decide if your company can benefit from adding relational database technology to your current capabilities.

This paper reports on the growing body of knowledge about relational technology. It begins by reviewing the challenges facing the MIS organization and the motivation for relational technology. It then briefly describes the history of relational technology and defines the basic terminology used in the relational approach. This will be followed by an examination of the productivity features of the relational approach and why it should be seen as a complement rather than a replacement for existing network databases such as the IMAGE data base management system. Typical application areas where the relational approach can be very effective will also be surveyed. Finally, a checklist will be reviewed that will help the audience determine if, indeed, they really can benefit from using a relational database.

INTRODUCTION

THE CHALLENGES FACING MIS

The MIS manager is facing many challenges in today's modern information systems organization. The backlog of applications waiting to be developed is one of key challenges concerning MIS. In most medium to large installations the backlog of applications waiting to be developed is anywhere from two to five years. This estimate doesn't include the "invisible backlog," the needed applications which aren't even requested because of the current known backlog. Software costs are increasing because people costs are going up and because of the shortage of skilled EDP specialists. The database administrator typically uses nonrelational databases where a great deal of time is spent predefining data relationships only to find that the users data requirements are changing dynamically. These changes in user requirements cause modifications to the database structure and, in many cases, the associated application programs.

The application programmer is spending a significant amount of time developing applications using these non-relational databases, which require traversing or navigating the data base. This results in excessive application development time. Because the users' requirements change dynamically, a great deal of time is spent maintaining applications. The programmer is also frequently restricted by the data structures in the database, adding to the complexity of accessing data.

End users or business professionals are frustrated by the limited access to information that they know exists somewhere in the database. Their business environment is changing dynamically, and they feel MIS should keep up with these changes. They find that the applications are inflexible, due to the pre-defined relationships designed into the data base. They also lack powerful inquiry facilities to aid in the decision-making process, which would allow them to ask anything about any data residing in that database.

THE MOTIVATION FOR RELATIONAL

Dr. Edgar F. Codd, considered to be the originator of the relational model for databases , noted when presented the 1981 ACM Turing Award that the most important motivation for the research work resulting in the relational model was the objective of providing a sharp and clear boundary between the logical and physical aspects of data base management (including data base design, data retrieval, and data manipulation). This is called the data independence objective.

A second objective was to make the model structurally simple, so that all kinds of users and programmers could have a common understanding of the data, and could therefore communicate with one another about the database. This is called the communicability objective.

A third objective was to introduce high-level language concepts to enable users to express operations on large chunks of information at a time. This entailed providing a foundation for set-oriented processing (i.e., the ability to express in a single statement the processing of multiple sets of records at a time). This is called the set-processing objective.

Another primary motivation for development of the relational model has been to make data access more flexible. Because there are no pointers embedded with the data, the relational programmer does not have to be concerned about following pre-defined access paths or navigating the database, which force him to think and code at a needlessly low level of structural detail.

THE RELATIONAL DATA MODEL: A BRIEF HISTORY

In 1970, Dr. Codd published an article in the Communications of the ACM entitled "A Relational Model of Data for Large Shared Data Banks." This classic paper marks the "birth" of the relational model. Dr. Codd was the first to inject mathematical principles and rigor into the study of database management.

By the mid 70's, there were two database prototypes being developed. IBM was behind a project called "System R," and there was another relational database being developed at the University of California, Berkeley, called INGRES. It was late 1979 before the first commercially available relational database, called ORACLE, arrived in the marketplace from ORACLE Corporation. ORACLE is also an implementation based on "System R". In 1981 Relational Technology Inc. introduced INGRES which was a different implementation based on the research done at Berkeley. Today there are several additional advanced relational products available, such as HPSQL from Hewlett-Packard, SQL/DS and DB2 from IBM, Rdb from Digital Equipment Corporation and SUPRA from CINCOM. There are additional products sometimes referred to as "born again" relational databases, such as IDMS/R from Cullinet, DATACOM/DB from Applied Data Research and ADABAS from Software AG, to name a few.

RELATIONAL DATABASE DEFINED

The relational database model is the easiest to understand - at least at the most basic level. In this model, data are represented as a table, with each horizontal row representing a record and each vertical column representing one of the attributes, or fields, of the record. Users find it natural to organize and manipulate data stored in tables, having long familiarity with tables dating from elementary school.

The Table, or two dimensional array, in a "true" relational database is subject to some special constraints. First, no row can exactly duplicate any other row. (If it did, one of the rows would be unnecessary). Second, there must be an entry in at least one column or combination of columns that is unique for each row; the column heading for this column, or group of columns, is the "key" that identifies the table and serves as a marker for search operations. Third, there must be one and only one entry in each rowcolumn cell.

A fourth requirement, that the rows be in no particular order, is both a strength and a weakness of the relational model. Adding a new record can be thought of as adding a row at the bottom of the table; hence there is no need to squeeze a new row in between preexisting rows as in other database structures. However, to find a particular row, the entire table may have to be searched.

There are three kinds of tables in the relational model: base tables, views, and result tables. A base table is named, defined in detail, filled with data, and is more or less a permanent structure in the database.

A view can be seen as a "window" into one or more tables. It consists of a row and/or column subset of one or more base tables. Data is not stored in a view, so a view is often referred to as a logical or virtual table. Only the definition of a view is stored in the database, and that view definition is then invoked whenever the view is referenced in a command. Views are convenient for limiting the picture a user or program has of the data, thereby simplifying both data security and data access.

A result table contains the data that results from a retrieval request. It has no name and generally has a brief existence. This kind of table is not stored in the database, but can be directed to an output device.

THE RELATIONAL LANGUAGE

The defacto industry standard language for relational databases is SQL. SQL is pronounced "SEQUEL" and stands for Structured Query Language. This name is deceiving in that it only describes one facet of SQL's capabilities. In addition to the inquiry or data retrieval operations, SQL also includes all the commands needed for data manipulation. The user only needs to learn four commands to handle all data retrieval and manipulation of a relational database. These four commands are: SELECT, UPDATE, DELETE and INSERT.

The relational model uses three primary operations to retrieve records from one or more tables: select, project and join. These operations are based on the mathematical theories that underlie relational technology, and they all use the same command, SELECT. The select operation retrieves a subset of rows, that meet certain criteria, from a table. The project operation retrieves specific columns from a table. The join operation combines data from two or more tables by matching values in one table against values in the other tables. For all rows that contain matching values, a result row is created by combining the columns from the tables, eliminating redundant columns.

The basic form of the SELECT command is:

```
SELECT      some data (column names)
FROM        some place (table names)
WHERE       certain conditions (if any) are to be met.
```

In some instances WHERE may not be neccessary. Around this SELECT..FROM..WHERE structure, the user can place other SQL commands in order to express the many powerful operations of the language.

In all uses of SQL, the user does not have to be concerned with how the system should get the data. Rather, the user tells the system what data is needed. This means that the user only needs to know the meaning of the data, not its physical representation, and this feature can relieve the user from many of the complexities of data access.

The data manipulation operations include UPDATE, DELETE and INSERT. The UPDATE command changes data values in all rows that meet the WHERE qualification. The DELETE command deletes all rows that meet the WHERE qualification and the INSERT command adds new rows to a table.

When retrieving data in application programs, it is important to remember that SQL retrieves sets of data rather than individual records and consequently requires different programming techniques. There are two options for presenting selected data to programs. If an array is established in the program, a BULK SELECT can retrieve the entire set of qualifying rows and store them in the array for programmatic processing. Alternatively, it is possible to activate a cursor that will present rows to programs one at a time.

SQL has a set of built-in, aggregate functions. Some of the functions available are COUNT, SUM, AVERAGE, MINIMUM, and MAXIMUM. They operate on a collection of values and produce a single value.

In addition to commands for data retrieval and modification, SQL also includes commands for defining all database objects. The data definition commands are CREATE, ALTER and DROP. The CREATE command is used to create base tables and views. The ALTER command provides for the expansion of existing tables and the DROP command deletes a view. One of the most powerful features of SQL is its dynamic definition capability. This function allows the user to add columns, tables and views to the database without unloading and reloading existing data or changing any current programs. More importantly, these changes can be made while the databases are in use.

PRODUCTIVITY FEATURES OF USING RELATIONAL TECHNOLOGY

Relational technology is one very important tool that can contribute to making data processing professionals more productive. The programmer can benefit from a facility called interactive program development, which allows the development and debugging of SQL commands and then permits the moving of those same commands into the application programs. It is convenient and easy to set up test databases interactively and then to confirm the effect of a program on the database. All of these characteristics make SQL a powerful prototyping tool. The on-line facilities of SQL can be used to create prototype tables loaded with sample or production data. On-line queries can easily be written to demonstrate application usage. End users can see the proposed scheme in operation prior to formal application development. In this prototype approach, people-time and computer-time are saved while design flaws are easily corrected early in development.

The database administrator profits from the productivity features already described for programmers. The database administrator has a great deal of freedom in structuring the database, since it is unneccessary to predict all future access paths at design time. Instead, the DBA can concentrate on specific data requirements of the user. Nonrelational models, on the other hand, require all relationships be pre-defined, which adds to the complexity of the application and lengthens development time.

Additional productivity features for the database administrator include the capability to modify tables without affecting existing programs and the capability to dynamically allocate additional space while the database is still in use. SQL goes far beyond many database management systems in the degree of protection that it provides for data. Views make it possible to narrow access privileges down to a single field. Users can even be limited to summary data. Protection can be specified for database, system catalog, tables, views, columns, rows and fields. It is also possible to restrict access to a subset of commands. These access privileges can be changed dynamically, as the need arises.

In many installations, the key to overall productivity is the ability of data processing to offload the appropriate portions of the development and maintenance to the end user. The flexible design approach of relational databases allows an application to be designed with the end user's requirements in mind. This could enable the DP professional to implement an application up to the point where the end user could create and execute his own queries, thereby expanding the application on his own and reducing his dependence on the data processing department. Through SQL, the end user is provided with extremely flexible access and simple but powerful commands.

RELATIONAL AND NONRELATIONAL: COMPLEMENTARY TECHNOLOGIES

Within a data processing department already using a well-established nonrelational DBMS, what role can relational technology be expected to play? We know that DP will not automatically drop everything and go to relational database technology. Rather, relational technology should be seen as a complement rather than a replacement for nonrelational database systems. Both approaches offer a host of benefits, and most applications can be implemented with either of the two.

The relational approach is preferred when the application has a large number of data relationships or when the data relationships are unknown or changing dynamically. The relational approach provides the needed flexibility to establish relationships at the time of inquiry, not when the database is designed. If the application has unknown or incomplete data specifications, which is usually the case in a prototyping environment, then a relational system may be preferable. If the application requires a quick turnaround, the quick design and implementation capabilities of a relational database can be important. The ability to handle ad hoc requests is a definite strength of the relational model as is the ability to extract data for use in a modeling, forecasting, or analytical framework.

The nonrelational approach is preferred for high-volume, on-line transaction processing applications where performance is the most critical requirement.

CHOOSING THE RIGHT TECHNOLOGY

The choice of the "correct" database management system must be based on the environment in which the database will be used and on the needs of the particular application. The key feature of relational technology is that it allows for maximum flexibility, and will probably be the choice for many new applications. On the other hand, nonrelational systems may continue to be preferrable for very stable or structured applications in which data manipulation requirements are highly predictable, and high transaction throughput is important.

The optimum approach for many MIS departments will be to use the relational system concurrently with the existing nonrelational system, matching the appropriate technology to the application. The only problem with such an approach is that the data for an application developed in one technology may sometimes be needed by applications developed in the other technology. Data may be "locked out" from an application that needs it, or users might be tempted to duplicate the data, maintaining both copies. The most desirable solution would obviously be to provide both relational and nonrelational access to a single database. This capability will be available with HP's ALLBASE.

RELATIONAL TECHNOLOGY CONSIDERATIONS

There are several things to consider when making the decision to go to a relational database environment. The additional resources usually required to support this technology could significantly impact your system . For example, the intelligence built into the software and the dynamic capabilities of the relational approach usually require additional CPU cycles and memory.

Performance is usually a factor when considering the relational approach and often depends on the maturity of the optimizer software which is built into the relational DBMS. The Data Base Administrator is very important when considering relational and plays a major role in monitoring and improving performance by creating and dropping indexes when neccessary. The DBA can also elect to use "clustering" or keeping "like data" together which affects performance by reducing the number of times a disc is accessed.

The command driven nature of SQL may be difficult for some users because they usually have to know the names of the tables and data fields in order to properly construct a SQL command and may prefer a much more "friendly" menu-driven interface. The SQL user must also know the beginning and end of transactions that modify the database and when to "commit work" against that database.

Security of the data resources is usually very important, and the DBA has the capability to implement some very comprehensive security schemes. In addition, to ensure data integrity, logging transactions is mandatory and the user has no way of turning logging off.

If your organization currently has SQL users in an IBM environment they will find little difference in a Hewlett-Packard SQL environment. The user and programmer interface is essentially the same; however, there are some Data Base Administrator functions which are system dependent.

Future releases of HPSQL will work with 4th generation languages and the System Dictionary. In addition, an easy-to-use menu-driven report writer for HPSQL end users and programmers will soon be available.

RELATIONAL APPLICATIONS

There are many application areas - particularly those involving user analysis, reporting, and planning - where the very nature of the application is constantly changing. Some typical application areas are:

* Financial
  - Budget analysis
  - Profit and Loss
  - Risk assessment

* Inventory
  - Vendor performance
  - Buyer performance

* Marketing and sales
  - Tracking and analysis
  - Forecasting

* Personnel
  - Compliance
  - Skills and job tracking

* Project management
  - Checkpoint/milestone progress
  - Development and test status

* EDP auditing
  - Data verification
  - Installation configuration

* Government/education/health
  - Crime and traffic analysis
  - Admissions/recruiting/research
  - Medical data analysis

These applications typify instances where it is of primary importance to establish interrelationships within the database and to define new tables.

CHECKLIST FOR DECIDING WHETHER OR NOT YOU NEED A RELATIONAL DATABASE

Note: If you answer yes to any of the following questions, you should seriously consider taking advantage of relational technology.

1. Does your company have an excessive backlog of applications to be developed, including an invisible backlog?

2. Is your company spending too much money developing applications due to the complexities of using nonrelational systems?

3. Are your programmers spending too much time maintaining applications caused by changing data requirements or relationships?

4. Are your programmers spending an excessive amount of time writing code to navigate through nonrelational databases?

5. Is the nature of your applications constantly changing?

6. Do your users' requirements for information change dynamically?

7. Do your users feel restricted by a nonrelational database?

8. Would your users find it natural to organize and manipulate data in tables?

9. Do your users currently use LOTUS 1-2-3 or spreadsheets?

10. Is your company moving towards a distributed database environment?

SUMMARY

Relational technology can have a profound effect on the way organizations operate. In short, the use of relational databases, within the correct DP environment, can help turn the computer into the effective tool most managers need to run their organizations successfully. The following conclusions deal with relational database technology.

* Relational concepts are easy to understand and use.
* SQL is a multifunctional language.
  - Database definition and creation
  - Data retrieval
  - Data manipulation
  - Authorization and security
  - Transaction management and recovery
  - Database environment management and restructuring
  - Interactive and programmatic use
* SQL allows you to specify which information you want - not how to retrieve it.
* SQL increases programmer productivity and lifts programming to the level of problem solving.
* Data independence is ensured and minimizes maintenance of programs
* Data access is automatically optimized as DB structure changes.
* The DBA has unprecedented power and control over the database.
* New systems are implemented much faster.
* Relational databases provide a cost effective powerful solution.

It is to the advantage of most data processing management to learn to use this technology creatively and to manage it effectively. The bottom line is that RELATIONAL DATABASE TECHNOLOGY IS HERE TO STAY!

REFERENCES

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM, 13 6,(June 1970),pp. 377-387.

Codd, E.F.,"Relational Database: A Practical Foundation for Productivity," CACM, 25 2,(February 1982),pp. 109-117.

Date, C.J., An Introduction to Database Systems, Addison-Wesley, 1977.

Date, C.J., An Introduction to Database Systems Vol II, Addison-Wesley, 1983.

Schussel, George,"Relational Database-Management Concepts, "Proceedings 1986 Database and Fourth/Fifth Generation Language Symposium,NY,NY, June 8-12,1986.

_____,Relational Technology: A Productivity Solution, Hewlett-Packard Co., Computer Systems Division, Cupertino, Ca., 5954-6676, January 1986.

_____,SQL/Data System for VSE: A Relational Data System for Application Development, IBM Corp. Data Processing Division, White Plains, N.Y.,G320-6590,Feb 1981.

HOW TO BUILD A DISTRIBUTED M.I.S. SYSTEM

Roger W. Lawson

Proactive Systems
Box 1425
Berkley
Michigan 48072

INTRODUCTION

Management information systems (M.I.S) have historically been seen as centralised systems. The typical large company organisation structure is a hierarchy with a peak at head office. Therefore computer systems to support this structure have been concerned with the consolidation of locally collected data towards the centre and the distribution of information from the centre. In the past this often resulted in a naturally centralised approach to the provision of data processing resources. However this strategy is not only inefficient and inflexible but is also needlessly expensive. I will attempt to show how such systems can be built using networks of minicomputers (eg. HP3000s) and discuss the practical problems that have to be overcome.

THE REQUIREMENTS

To support any complex data processing system or M.I.S. system that comprises multiple applications you need the following capabilities:

- The ability to share data between applications.

- The ability to pass transaction data from one application (or process) to another.

- The ability for users to access data from multiple data files easily and transparently.

Now if all the systems are running on the same computer, this is easy. However the centralised approach has many disadvantages. Some of these are:

- You are placing all your eggs in one basket. If the central computer stops then all application systems stop. Even a short interruption is very costly and the company as a whole may be vulnerable if a real disaster such as a fire occurs.

- The data communication costs are very high if the users are geographically dispersed. If the computer is located in Los Angeles but some of the users are in New York it is a very expensive solution.

- It is inflexible as an upgrade of processing power may only be possible in steps (also there tend to be upper limits on such equipment as is so with the power of the HP3000 computer range).

- Groschs Law may no longer apply. Multiple small computers can give you cheaper power than one large one.

THE PROBLEMS OF DISTRIBUTED SYSTEMS

If you spread applications over multiple computers then to meet the requirements mentioned above is much more difficult. Let's take each in turn:

- You need to be able to share data between computers that are geographically remote (although probably linked by a communication system such as DS/3000 even though this may be subject to frequent failure or interruption). For example you may be running a sales order processing system on one computer and a production/inventory system on another computer - they both need to share (and update) the same stock availability data. Although DS/3000 allows access to remote data bases it provides no facilities to logically link two or more data bases on seperate computers.

- Moving transaction data from one system to another is much more difficult in a distributed system. An example is where a sales transaction needs to reduce a stock balance which is held on a local computer plus create an accounts ledger entry on another central computer. DS/3000 provides the facility to easily copy a file in "batch" mode but building a real time processing network with automatic recovery in case of a failure is another matter.

- User access to the data for both read and update purposes introduces technical problems. For example on an HP3000 if you have more than a few remote DS sessions then performance tends to be very poor. Also if you have a reasonably complex network (ie. more than 2 computers) and you have lots of remote data base access then you are very vulnerable to failure of any one node or any communication link. Obviously the more computers you have then the higher the risk you run of one being out of action.

OTHER REQUIREMENTS

Other common requirements that are not covered easily are:

- Semi-static data on one computer needs to be reflected on other computers in the network. For example a price list that is maintained centrally needs to be distributed around the network. Note that in this case because the data does not change very frequently and is not large in volume it is more cost effective for each computer to have a local copy. The trade off is the cost of disc space against the communication and performance cost of remote data base access.

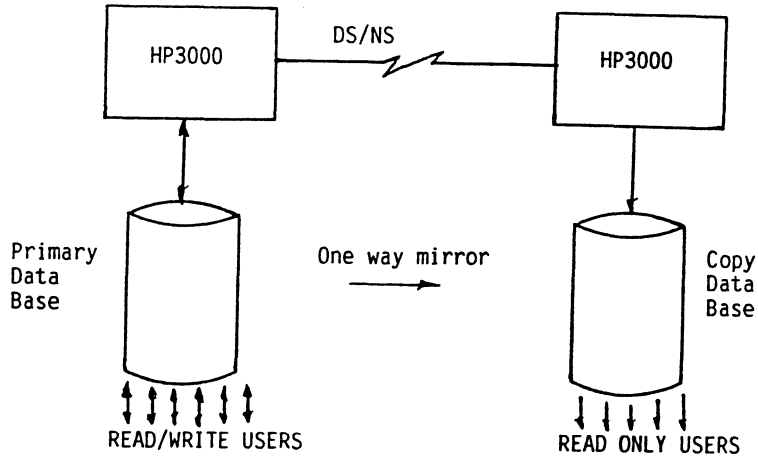
The "local copy" approach also makes each local system less vulnerable to network failure. Now you could do this by passing transaction data around the network and having some local processing code on each system, but all you really need is some system software which can be instructed with a command that says "keep the price data sets on computers 2, 3, 4 etc. automatically the same as the price data set on computer 1".

- Data consolidation is often required. For example sales transactions that are processed by each local computer need to be collected and merged in real time into a single data set on a central computer.

A SOLUTION

Now a couple of years ago my company came up with a solution to the above problems which is a software product called BACKCHAT. The original germination of the product stemmed from both my and a colleagues experience in running multiple HP3000s as users (for example the company I worked for as DPM had over a dozen linked machines). Originally BACKCHAT was aimed solely at providing a real time copy of an IMAGE data base on a second computer by "mirroring" the data base. This application is very similar in purpose and mode of operation to HPs SILHOUETTE product (BACKCHAT is an alternative to that but with many more facilities). This mode of operation is represented in Figure A below.

Figure A



In this case the copy data base can be read by users on the secondary HP3000 but cannot be updated. This system provides:

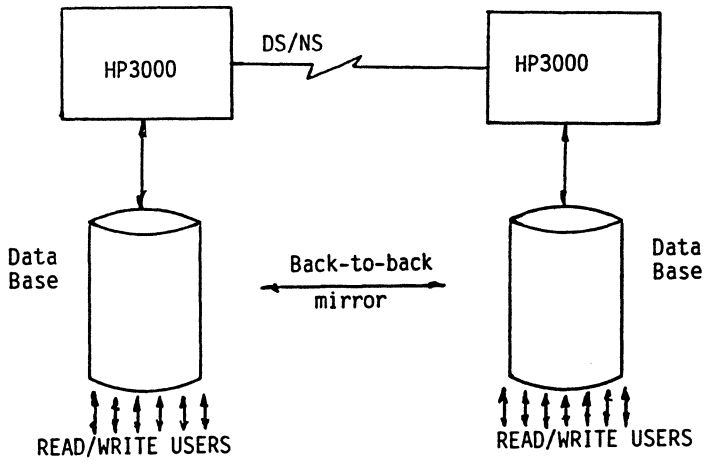
1. A disaster protection system (users on the primary system can be switched to the secondary if the primary fails).

2. Load spreading as reporting/enquiry users can be off loaded to the second computer.

3. Concurrent back-up and 24-hour availability as tape stores can be done on the secondary without stopping access to the data base on the primary.

After releasing the product we were approached by a potential client who was not only interested in the disaster protection capability but who also had a number of other requirements to enable him to build a distributed system spanning England, France, Germany and Australia. One of his needs was to be able to logically share a data base (or data set) with updating on both systems. This is represented in Figure B.

Figure B



So we enhanced the product to provide this functionality (like many advances in the state of the art of computing the development grew out of close interaction between a software company and a user). With the new capabilities we can supply all the distributed data base requirements mentioned above.

TECHNICAL DESCRIPTION

BACKCHAT works by using the IMAGE logging system to pick up data base changes, passes the transactions to a remote processor and applies them to the remote data base. With concurrent updating as shown in Figure B there are effectively two of these processes in operation (one in each direction) so that the data bases can be viewed as being mirrored "back-to-back". There is special logic incorporated to stop transactions echoing backwards and forwards for ever. There is also special logic to cope with record relocations (record numbers of IMAGE records in detail data sets changing from one data base to another).

There is a lot of control and configuration logic associated with controlling:

- Multiple data bases concurrently.

- Multiple remote connections.

- Selective parts of data base (eg. data sets).

- Restart and recovery from failure.

- Simple operator control from one location.

Note that BACKCHAT uses the IMAGE logging system for two reasons:

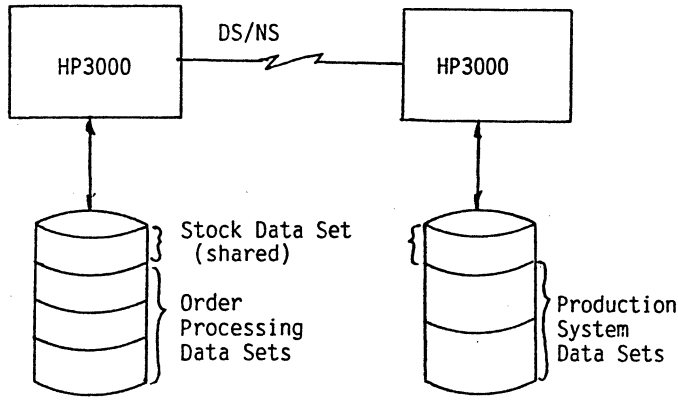
1. It is the most efficient way of picking up data base changes, i.e. has least performance impact.

2. It contains logical transaction definitions which may need to be used in recovery situations.

Incidentally BACKCHAT does not use privileged mode.

With this kind of software one can easily "share" information across a network without special programming. For example with a simple configuration dialogue it is possible to set up sharing of stock information as in Figure C below.

Figure C



As you can see the data bases do not need to be similar except for the data set that is to be shared.

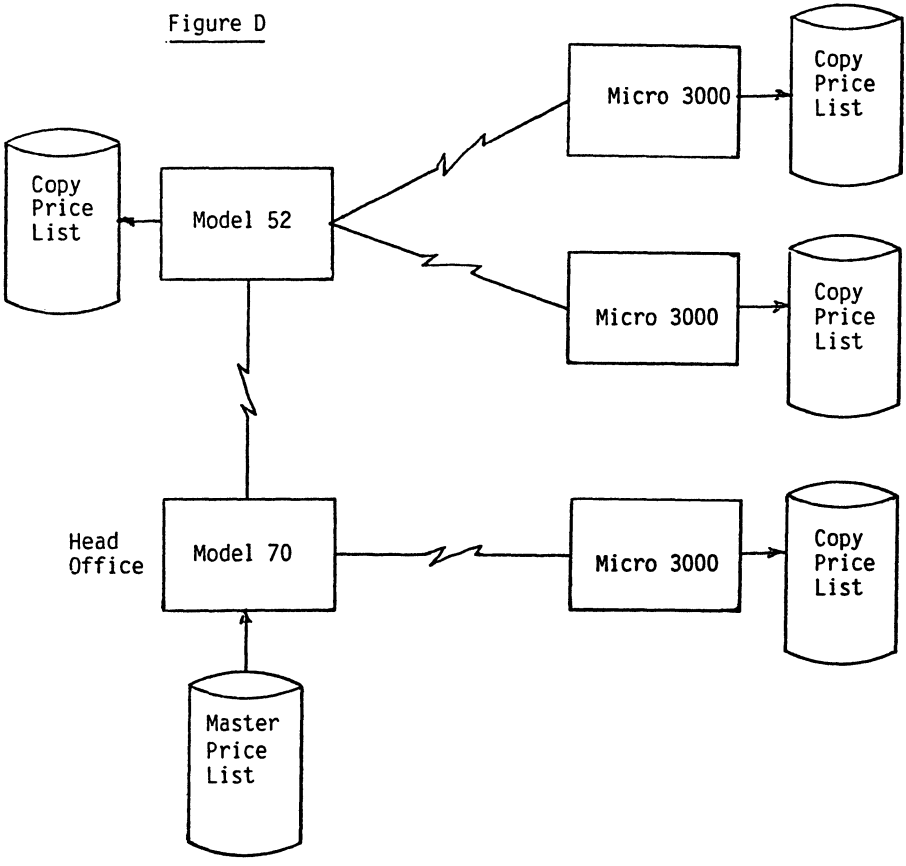
CONCURRENT UPDATING

Now the obvious question with this design is "what limitations are there on concurrent updating". Firstly if the data sets within a data base are only updated on one system then no potential conflicts arise. Also the software can handle updates to the same data set so long as the particular records being updated are different. However because it is not practical to impose a global lock (which would reduce the tolerance to network failure in any case) the concurrent updating of the same record has to be considered with care. For example if a record was deleted on one system at the same time as it was updated on the other then when the update arrived on the first system (which may be some time later depending on configuration etc) the record would have disappeared. However with suitable application design it is possible to avoid these problems (eg. flag the record for later deletion in this example). Although it may not be possible to take an existing application running on a single computer and chop it in two without application changes, there are effectively no significant limits on what you can achieve.

MORE EXAMPLES

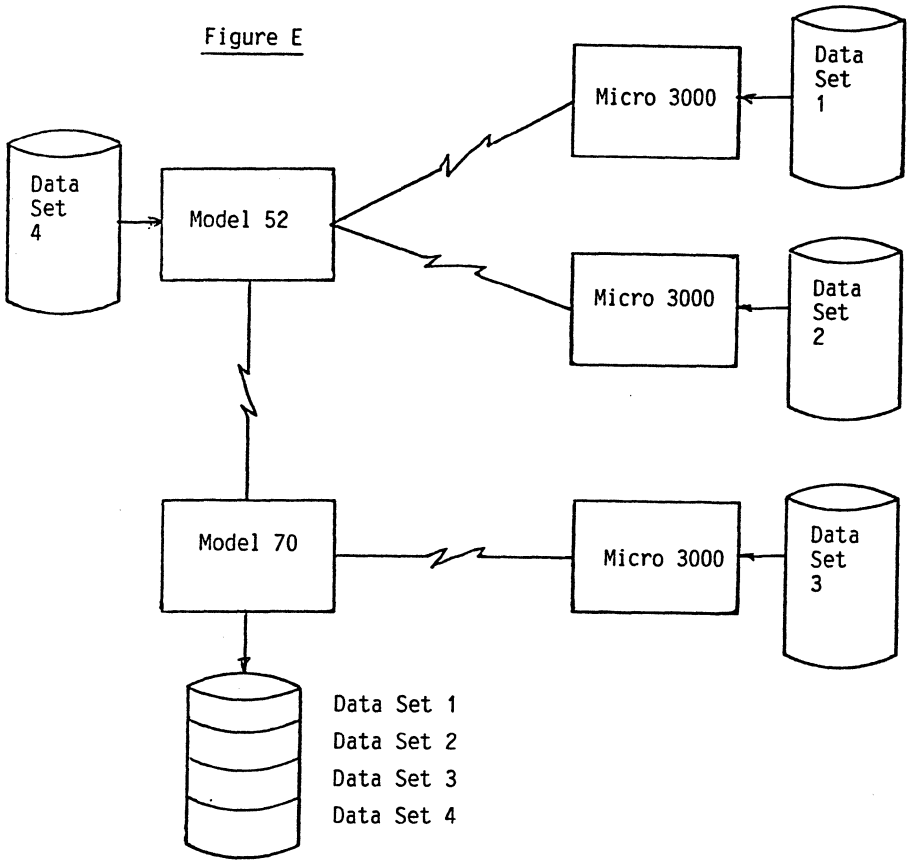
Figures D and E below show how easy it is to provide the shared reference information and data consolidation facilities.

Figure D



In this example head office maintain the master price list on their computer and BACKCHAT automatically maintains copies on the other computers (the intermediate copy on the Model 52 could be dispensed with if not required).

Figure E



In this example branch sales are automatically passed from each local system as they are collected to the central system. They are then automatically consolidated into a single data set on the central system. Note that BACKCHAT also contains utilities to help set up the merged data set initially. In this example transactions are effectively transmitted through the network for further processing at another location.

TOLERANCE TO FAILURE

Because BACKCHAT processes and manipulates log records in files on both local and remote systems these files effectively act as buffers in the network. This means that the seperate computer processors can operate asynchronously and, if a network failure occurs, transactions simply accummulate in the buffers until the relevant part of the network is restored (BACKCHAT can then automatically recover to the correct point). Whether it is communications line failure or computer failure, in neither case are other computers in the network significantly affected using this approach.

The software also contains such facilities as a roll back recovery module that is invoked in certain failure circumstances (it includes linked roll back of logical transactions that span multiple data bases by looking at the user PIN number to identify common usage).

CONCLUSION

I hope I have shown how it is possible to build distributed data base solutions now on HP3000 equipment using IMAGE. With the approach described all the requirements of distributed data processing can be easily met. It is certainly practical to build real integrated networks and do the computer processing in the locations best suited to minimise costs and maximise user convienience.

A PERFORMANCE COMPARISON OF HP3000 REPORT WRITERS

Roger W. Lawson

Proactive Systems
Box 1425
Berkley
Michigan 48072

Running reporting programs or procedures often consumes a high proportion of the cpu and IO resources of an HP3000 computer system. This paper supplies data on how the commonly used report writers compare on performance. Also it takes a specific look at HPs new Business Report Writer (BRW) and sees how it stands up against other report writers (Hewlett Packard have been promoting the new BRW product as the answer to all reporting requirements on HP3000 systems). Having recently had the opportunity to try it out (and compare it to other reporting products performance wise) I have collected some interesting data.

Let me first declare that as I work for a company that produces an HP3000 writing product (namely Q-GEN), I will not attempt to present a full review of BRW or the other products - it would take more time than I have available to perform a full evaluation of each. However BRW particularly interested me because a couple of years ago I did an evaluation of QUERY, INFORM/REPORT (from HPs RAPID package), QUIZ (from COGNOS), ASK from COGELOG and a COBOL report writer (Q-GEN from PROACTIVE SYSTEMS). The evaluation was purely a comparison of their run time performance ie. how much load they placed on an HP3000 system when producing a typical sample of reports. To produce the data each sample report was written in each reporting language (not a simple task incidentally as it means you have to learn all the report writers and have a copy of the software on your system). The report procedures were then run against a medium size data base on a Model 44. The resulting figures were published in the HP IUG journal and presented in a couple of talks - a summary of the data is shown in Table 1 overleaf.

Table 1

Relative Performance in CPU seconds (COBOL=1)

	COBOL/ Q-GEN	QUERY	REPORT	INFORM	ASK	QUIZ
Test 1	1	2.1	1.8	2.1	1.9	2.3
Test 2	1	2.1	2.0	2.1	1.9	2.4
Test 3	1	4.7	8.7	7.9	1.9	2.2
Test 4	1	5.0	N/A	N/A	2.0	N/A
Test 5	1	4.4	N/A	N/A	2.0	2.0

Notes. N/A= Not Available. Relative performance in elapsed times were very similar and are therefore not shown. The tests range in complexity from very simple (test 1) to relatively complex (test 5) - see end of article for test details.

The figures highlighted what many people already knew. Namely that QUERY is a real machine killer, INFORM/REPORT are not much better, and that while QUIZ is better than QUERY it is still typically twice as slow as a reasonably well written COBOL program. Note that although claims have been made that INFORM/REPORT and QUIZ have been improved over the last couple of years since the original tests were done I doubt whether the figures would be significantly different today. Note also that the relative speed of the products does not vary much from one HP3000 model to another or between one configuration and another.

Now Hewlett Packard have been saying to some users that "BRW is not only very flexible but it can also be faster than COBOL" ie its the answer to the system managers prayers. To test this out I simply did some more comparative tests. I took two reports (one simple, one complex) and wrote them in QUERY plus rewrote them using BRW - I also converted the QUERY procedure to a COBOL program using Q-GEN. The three versions were then run on one of our mailing list data bases on our Model 37. The results are shown in Table 2 overleaf.

Table 2

Relative Performance in CPU seconds (COBOL=1)

	COBOL/ Q-GEN	QUERY	BRW
Test 6	1	2.2	2.4
Test 7	1	2.9	1.9

See end of article for test details. Relative performance on elapsed times were again very similar.

Well the first surprise is that on a very simple report, BRW is slower than QUERY! I didn't believe this when I first saw the figures so I reran the test with a similar result. Now BRW has one useful feature in that it gives you a breakdown on $STDLIST of where it is spending the time when running a report - it appears that BRW is wasting a lot of needless time writing to and reading from a work file.

At least on a more complex report BRW seems to be a lot better than QUERY - it is probably more comparable in performance to QUIZ but it is still nowhere near a COBOL program (even one generated rather than hand coded).

Other comments - BRW has a very sophisticated, menu-driven report definition system with a compiler. I would have done more testing but the other users on our Model 37 complained that when I was running the report definition system, nobody else got a decent terminal response - as it takes a long time to step through the menus to produce a report this is not going to be a helpful feature on smaller machines (BRW seems to eat up a lot of memory). However if your other users can put up with it then BRW looks very powerful - but without doing the 4 day H/P training course I found it difficult to understand fully how to use the system from reading the reference manual. Even getting a copy of the manual from H/P proved difficult even though I was happy to pay for it - I had to borrow one from another user. BRW was written in Germany - it probably shows that in the number and comprehensive of the products facilities. One oddity was that I found I had to set up a dictionary for the test data base using DICTIONARY/3000. There is then a utility provided that is used to create an MPE file containing an extract of the dictionary - you can then throw away the dictionary (maybe H/P should provide short term rentals of DICTIONARY/3000 so that you don't need to buy it!).

If you are looking for a product as comprehensive as BRW then maybe the old RPG product is worth a look - it is certainly as flexible as BRW, is easier to learn, is industry standard and is well proven.

Note that BRW has many good points but end-user report writer it is definitely not. Non d.p. staff will even have difficulty understanding the HELP screens within BRW.

CONCLUSION

As an alternative to INFORM/REPORT for sophisticated users who wish to use a data dictionary (and don't mind the associated cost) then BRW warrants further evaluation. As an alternative to the free QUERY product (or other third party products) then BRW does not look so exciting. The effort to learn the new product (apart from the time to rewrite old QUERY procedures) is too high.

However QUERY is certainly slow - partly because of the interpreted nature of the product and partly because of the design of the record handling. Also it has certain limitations (such as lack of conditional printing and limits on the numbers of statements). To overcome these problems we developed a couple of years ago a compiler for the QUERY language which is called Q-GEN - it actually generates COBOL source code which it then links and compiles for you (or you can get the COBOL source and play about with it yourself). We also support extensions to the QUERY language such as an "IF" statement to do conditional processing, much higher statement limits, European date format etc.

This approach gives you much improved performance as you can see from the figures above - for example one of the original applications was to reduce an overnight reporting run from 15 hours (which meant the old version often did not finish until the next morning) to about 5 hours.

It also gives you total flexibility because if the QUERY extensions provided in Q-GEN are not enough you can always modify the COBOL code.

Even Hewlett Packard like the product - so far they are purchased about 30 copies for use in their own offices in different parts of the world.

TEST DETAILS

Tests 1 to 5 were run primarily on a Model 44 (with some repetition on a Series III). Tests 6 and 7 were run on a one megabyte Model 37 with two 7914 disc drives (running U-MIT and TurboIMAGE). All tests were run in job mode with no other jobs or users on the system.

Test 1 was a serial search of a detail dataset to retrieve 7000 records out of 59000. Report was an unsorted list of the records with no totalling and limited editing.

Test 2 was the same as Test 1 except that the records were sorted and only totals were printed (one level of sort).

Test 3 was the same as Test 2 except that a single data item from another data set was printed in the report total lines (item obtained from another detail data set via a common master).

Test 4 was a serial search of a detail data set to retrieve 13600 records out of 92000 records. Report records were sorted and totalled at one level with an item retrieved from another data set as in Test 3.

Test 5 was a serial search from a detail data set to retrive 6600 records out of 59000. Report records sorted and totalled and data from a linked master data set included in the report totals.

Test 6 was a serial read of 2731 records from a detail data set (out of 2732 records). The report was a simple list of two items from the data set with no sorting or totalling.

Test 7 was a serial retrieval from two detail data sets linked by a common automatic master (2700 records in each data set). 1870 records were selected for printing with sorting at 3 levels plus group totalling. Final report total, page numbering, heading lines etc also incorporated.

User-Sympathetic: The New Standard for User Interfaces

by

Clifford W. Lazar
President,
SystemsExpress
15015 Ventura Blvd. Sherman Oaks, CA 91403
818/907-4800

Originally Published in SuperGroup

Most of us feel we are writing user-friendly systems, or would, if we had the time. As a result, what I am going to outline here may likely re-inforce your view of your work. How great you are.

What was Man's Earliest Important Invention?_____

As an exercise don't read the next paragraph until you write down what you think was Man's most important early invention. First of all, an invention is something that doesn't already exist in nature or wasn't previously created by someone else. Fire occurs naturally in nature in the form of lightning-started prairie fires and Biblical burning bushes. O.K., write your answer in the blank.

Did you write down "the wheel"? Well, you are wrong. The wheel occurs naturally in nature as round stones, logs, and watermelons. If you step on any one of these on a mountain path you will go over the edge.

Still you visualize a Basbylonian with a solid wooden cart wheel.

The invention, however, was the axle. You remember that the slaves of the Egyptians moved big blocks of granite with round logs (rollers or elongated wheels) by having the slaves drag the last roller wheel into the front while the pushers and pullers kept the block moving forward. This high inertia system with a high level of human involvement resulted in a lot of fatal accidents.

The value of the axle is that it brings the wheel along with the cart or chariot. A wheel without an axle has very little value except to act as a large paper weight, a saloon chandelier or a base for a teather ball pole.

Let me beat the analogy to death a little bit more. The ruling class of Egypt, the military, used axles on their chariots. The captive audience just used the wheels. People who use axles, use them because time is important and it would be inefficient to have to drag the rollers up to the front every time.

The people to whom we give the rollers are captives whose time we don't value and who have no other choice.

Much of the software offered on the HP 3000 is like rollers. The function is fulfilled but it is a drag for the user.

Now if anyone accuses you of re-inventing the wheel, you have two good reasons for spilling coffee on his shoes.

User Friendly vs Programmer Friendly

There is a confusion about the meaning of User Fiendly. Programmer Friendly is NOT User Friendly.

Programmers like puzzles and like to hack. Users get frustrated at the least buckishness of the computer and tend to give up.

Three Levels of User Friendly

User Friendly, as it is currently being misused, actually has three levels:

- User Hostile
- Programmer Competent
- User Sympathetic

User Hostile

User Hostile systems can be best understood by citing examples. The catagories of user hostile are:

- Sudden Death
- Mirages
- Quicksand
- Tar Babies

Sudden Death

Sudden death means that a small mistake can cost you hours of invested time when your file disappears or your database gets corrupted. In the CP/M version of WordStar on the HP 120/125 series, if you attempted to save a file to a protected disk, WordStar aborted and you lost your file. In its earliest version, Math/3000, the first spread sheet program for the 3000, would die if you tried to take the square root of a negative number. Hours worth of work would disappear. When the failure was reported to Mike Atlas, the creator of Math/3000, he fixed it right away. Math/3000 is now a great time and money-saving product.

Mirages

Mirages define apparant visions that are not actually there. The program doesn't do what the manual says it would do because important paramenters were left out of the explanation or the software was revised but the manual wasn't. Or worse it was assumed that the user knew something he didn't know.

Mirages require that the user struggle through a shimmering desert of blunders, guesses, tries and indications until he finds out how the software really works.

Quicksand

Quicksand refers to the case where you make an error and are caught and can't extricate yourself so you sink below the surface and die.

In one case of quicksand, the programmer, acting cute, accepts your input character-by-character so you can't back space to correct your mistakes even if you see the error before the machine acts. You can't work around the error and so eventually the only choice is to abort and start over again, if you're not in an OPTION NOBREAK.

Tar Babies

A tar baby occurs when you make a decision in good faith, discover it's a bad choice and can't back away. Like Br'er Rabbit, who just wanted to shake hands with the tar baby, your only choice becomes to leap into the briar patch. You may have made a reasonable choice at the time, but you can't extricate yourself without being covered in embarassement.

For example you discover that the design of your database structure is inefficient but you can't conveniently change it without purging and retyping your data and looking real dumb to your client. Especially if she has to retype it.

Programmer-Competent

Many programmers are Programmer-Competent. They're geniuses. They're bored at Mensa meetings. They have no sympathy for lesser mortals who can't memorize 500 page alphabetically organized manuals, type at 50 words per minute, without errors, or compose 50 character commands with mixed and inconsistent delimiters such as

```
:NEWACCT PAYROLL,MANAGER;CAP=IA,BA,PH;ACCESS=(R,X,L:ANY;W,A:AC)
```

Note: The delimiters are spaces, commas, semicolons, equal signs, colons and left and right parentheses. Seven different delimiters.

Programmer Competents also seek to maximize system functionality and options, thereby expanding the user manual so that 80% of the pages are irrelevant 99% of the time. Unfortunately, Competents want you to read all their options in alphabetical and then suboption order. They love all their children equally and therefore won't tell you which commands can be ignored.

User Friendly used to mean anything that wasn't IBM JCL. JCL was terrible. The manual was 1500 pages long. People who has suffered through JCL used to think that MPE was user friendly. It was -- seven years ago when most of the users were programmers. Now it is programmer competent.

Drowning in Online Help

The new trend is toward online help with megabytes of explanations and deeply embedded examples available at the press of "HELP STORE,EXAMPLE". The documentors feel there is safety in multiple screenfuls of characters so the explanations roll off the top of the screen at 9600 baud without even a "Press any key to continue" that you normally would get on any micro computer program. Even Query has "Press any key."

Systems written by competent programmers generally require week-long, $900 per person, training courses before you can start to use the product.

If a sex manual were written by HP it would be 780 pages long, in alphabetical order and you would be expected to read the whole book before you could get any hands-on experience.

Irreduceable Minimums

It's interesting to consider what might be the irreduceable minimum commands for any system.

You would need commands for the following:

File Handling

- Creating Files
- Loading Files
- Saving Files

Data Handling

- Adding Characters to the File
- Deleting Characters
- Moving the cursor (at least back and forth and possibly up and down)

Decision Automation

- Some kind of IF/THEN operator

These seven command types are included in 4GL's, editors, payroll programs, games and practically every other kind of software immaginable. In some cases the commands are defaulted, such as loading a game disk loads the file.

Programs that deal with numbers also may have calculation capability which ranges from two operands and an operator to full algebraic parsing.

The first computer that I worked on, the SWAC, had only 16 machine language commands, one of which was a no-op. At that time we felt that all that was really needed was a subtract with a conditional jump on negative result. The ultimate RISC machine.

The Fundamental Dilemma

The fewer commands that you have, the more commands you have to string together to accomplish a given task. The more commands that you have, the more commands you have to memorize.

Since Programmer-Competents have excellent memories they are prone to expect the same of users. Competent programmers expect a lot from users.

There is an alternative.

User Sympathetic

I feel there is a useful distinction between the meaninglessness of "user-friendly" and the implied style commitment of "user-sympathetic." By style commitment I mean that the programmer has a commitment to consider the user at every step and to sacrifice his own programmer convenience in the interest of the user.

It has been my experience that users aren't even appreciative of a user-sympathetic program unless they experienced a user-hostile version of it. Like exercising your civil rights without considering Tom Paine, Patric Henry or Martin Luther King.

Assumptions

User Sympathetic programs have certain assumptions in common:

- The user's time costs money
- The user can't remember anything
- The user only cares about his problem and not the solution process
- The user needs rapid feedback
- The user doesn't know and doesn't want to know MPE
- The user WILL make mistakes
- EFFECTIVELY, the user can't read!

The Fundamental Dictum

The fundamental dictum for user sympathetic programmers is that if they can write procedures that can do a function for a user in less time than the user can do it -- write the procedure! This is "AXLE MAKING".

The Game-theoretic Correlary is that if the procedure can probably save the user time--write the code! This means that if you can guess what file the user will want to use, then display the guessed file name and let him accept it with an ENTER or a RETURN or let him overwrite it.

Eighty percent of the time, users re-use their files. If they load them, they'll probably save them. If they edit them they'll probably compile, prep and save them. WordStar remembers the

last file used and you can invoke it with three keystrokes:

CTRL-R RETURN.

User sympathetic programs have little or no syntax and tiny or no manuals. :DBTranspose, my database extract and copy program, has a one word user manual: DBTRANS.

Tools of the User-Sympathetic Programmer

The user-sympathetic programmer seeks to minimize syntax, memorization and keystrokes. The alternatives to syntax are

- Menus and Softkeys
- Problem receptors

Menus and Softkeys

Menus are obvious and softkeys are dynamic horizontal menus. Memus shouldn't be just aids to memory. They should reduce keystrokes by knowing the structure of the options. Don't require that the user type the full name of the file or option. Let her select it by number or letter or let her tab to it and hit ENTER or RETURN.

Problem Receptors

Pascal, Fortran, Cobol, ADA, Basic, BAL and practically any language will solve nearly every immaginable problem. In order to write a simple program to open a file, read in the height, diameter, wall thickness and stress modulus of a tank and compute the strain at its base it is necessary that you read the entire manual of any of the languages mentioned.

In contast, if a program had already been written that prompted for the values, checked their ranges and warned the user if they were anomolous and eased making corrections, then this program would be a problem receptor.

A problem receptor knows what problem the user wishes to solve, and what terms he wants to use. It doesn't load him down with the optional capability of computing the red shift of receding galaxies if all he wants is to compute wall stress.

A problem receptor deals with finite possibilities within specified ranges. It asks "WHICH?" not "WHAT?"

If you buy a car you don't want to know how to turn brake drums. If you want to build a database system you don't need the option to compute tensile stress.

Tools for Problem Receptors

If you want to build problem receptors, your tools will include

- Prompts
- Error Management

Prompts

Prompts are more than just "Type File Name:". Prompts, in the larger sense, should tell the user

- where he is
- what input is required
- what are the options/range
- what are the possible negative results
- what to do next

A prompt is not just a single line. It is a component of an integrated whole screen.

Error Management

Users make errors. NASA administrators make errors, Presidents make errors. Good systems design assumes errors and manages them.

Coaching Little League Baseball is error management. The little tykes will make errors. The team that makes the least errors wins. At my old oil company we knew that we would drill dry holes. We felt that the company that drilled the shortest dry holes, while still finding oil, would make the most profit.

The clipper ship, Spirit of Baltimore, sank suddenly in an Atlantic storm. Unfortunately, the lifeboats were not readily accessible, not puncture proof, were not supplied with radar reflectors, were not supplied with enough food or shelter or water pumps for a reasonable expectation of survival in the ocean. That was not good error management.

Building systems without error correction and disaster recovery is not good error management. Building systems without good error management violates your fiduciary responsibility to your company's stockholders.

Avoiding Sudden Death

In user Sympathetic programs, sudden death should be impossible. Purging or overwriting of files should require comfirmation by the user. The warning should be distinct--possibly flashing and accompanied with a bell or two. "Overwritten" is too benign a phrase. The message should say:

"The data in your file XXXXX will be destroyed if you type 'Y'."

"Y" is a good choice because it is in the top center of the keyboard. Avoid using DEATH STAR keys that are on the bottom and the sides the keyboard where users might rest their hands. ENTER are keys placed in these dangerous positions so don't allow sudden deaths to be confirmed by simply pressing ENTER.

You might consider copying to-be-purged or overwritten files to father and grandfather files, if space permits.

Gang changes should allow for item-by-item confirmations. Cobol Accepts or Pascal Reads that are of fixed field length are dangerous because the user can't retract the last character. You make a mistake and the machine runs amok!

Fixups

If the user made some error that needs to be corrected, he should be told clearly

- what was wrong
- what to do to fix it
- the next key strokes required

The program should display error meaning and correction guidance and not expect you to look in the back of the manual.

It would be benefical if all the user's good choices could be saved so he could jump right back to the last good step without re-keing all his previous choices since the error.

MPE does this on a line-at-a-time basis for fixups with the REDO command. It fails to help you if there is a sequence of commands and only the last one reveals that the first command was bad syntax.

MPE often fails to give informative diagnostics and fixup instructions.

```
BAD SYNTAX; CIERROR=1234
```

is inadequate help when MPE potentially commands megabytes of meaningful messages it could send. My product :DBEXPRESS will tell the user what is wrong, how to fix it and what key strokes are necessary.

Key Issues in User Sympathetic Systems Design

When a system is designed to be user sympathetic the programmer should consider the following issues:

- Focusing
- Functional Subsets
- Heuristics
- Speed
- Standards

Focusing

Often, systems designers seek to solve all conceivable problems at the same time, with the same software. The result, like Lotus' Symphony, is a lot of compromises that don't include the best word processor, the best database manager or the best telecommunications package. The integration may be outstanding, but users may opt for specific packages and file transferrability.

Attempting to solve all the problems means that the system has to have a rich instruction set that the user must memorize. Focussing allows the program to better anticipate the wants of the user and speak in the user's vocabulary.

Elevator vs. Tree

Consider the difference between an elevator and a tree. You could climb the tree and get to any balcony on the front of a building on any of five floors. The different branches give you all the possible options. Because of the leaves, you have to memorize all the branch options before you start climbing to avoid getting lost.

Instead you opt for the elevator. Its simple. There are only five buttons. An elevator has a very limited vocabulary. Everyone understands it. After you ride the elevator with its five sysmbol vocabulary then you can walk down the hall and scan for the room number. That's simple too and everyone understands it.

If you are building a system with lots of options, group them and allow your user to go through successive simple menus, or if he has a good memory, jump to the option he wants.

Functional Subset

Consider WordStar. A trip to a good bookstore will reveal at least ten books costing between $14.95 and $21.95 on how to use WordStar. WordStar has over 500 commands and thousands of variations. It turns out that if you know 23 of the commands--that would fit on one piece of paper--you've got 95% of the power at your fingertips.

The concept is that you can give your user a functional subset of the system commands and he can get along fine--possibly being a little inefficient until he is comfortable with it.

If, when you were born, your parents gave you the Unabridged Oxford English Dictionary and told you to learn it before you could speak...

The average user, if you allow him to use MPE, needs only these commands:

```
:HELLO
:ABORT
:LISTF
:SHOWME
:FCOPY
:some UDC's for applications
:HELP
:BYE
```

All these commands could be summarized on a small reference card. Then, if the user wants to become a power MPE user let him go to a larger book that's organized in process, and not alphabetic order.

Heuristics

Since we are working with computers with very fast and very large memories, we should be willing to use those memories to be sympathetic to our user. Have the system memorize every choice the user makes and then, when the user cycles through the system again offer him the choices he made before. Let him accept them with ENTER or RETURN or overwrite them. Some companies call this AI, artifical intelligence, we don't. We call it AC, artifically clever.

Print Master, the slide generating program I use, remembers all the choices that I made. I can cycle to the next slide by just hitting RETURN six times without looking at the screen. If I want to change type fonts or layout options I can press the UP or DOWN arrows to highlight a different option and then press RETURN.

We are still haunted by the memories of small and slow computers. Like our grandmothers, who still save Christmas paper, we save memory space and machine cycles at the expense of our users.

Speed

Speed is of critical importance. Any delay of greater than two seconds will raise anxiety in the user. A programmer is likely to get up and go for a cup of coffee. There are some tricks to reduce the anxiety and the heart break of necessarily slow processes.

Avoid serial searches if possible. Train people to uses chains and hashes. Train your software to do the same. Don't default to serial search, even if it's simpler to code..

INFORM/3000 will allow a user to design a sorted report that will lock up his terminal for an hour or more. Warn the user of the implications of a 10,000 record sort. It's trivial to estimate the number of seeks and the sort time in terms of orders of magnitude. Display messages such as

```
This report will take over 30 minutes
Do you wish to stream the job offline so
you can still use your terminal?  Y or N:
```

VESOFT's MPEX is very good about offering stream options.

Before you start serial selecting all the records that are in Zip code 94010 out of a nationwide database, consider setting up automatic masters to states or area codes. The user might then select against the subset of area code 415. If you warn the user about the implication of giant serial searches she may think of alternatives.

If you are doomed to run a process that takes longer than five seconds, then display status information such as creeping periods or percentage completion or records processed. This will keep the user entertained and relieve her anxiety. Turbo Pascal counts lines as it compiles so you know when it will be done and how many lines your have.

Standards

To get out of Query you must type EXIT. The get out of EDITOR you must type E. To get out of Segmenter you must type QUIT. To get out of SPOOK you type E or Q, with differing results. To get out of MPE you must type BYE.

To get out of any part of :DBEXPRESS or FORMSPEC you must press f8.

Why can't all exits be standardized?

Good forms designers always had f5 and f6 mean PREV and NEXT. Some groups practice standardization and some don't That's bad.

Look at Phillipe Kahn, the father of Turbo Pascal. He built it with an editor that had the same commands as WordStar, the standard of editors. The result was that 10,000,000 people knew how his editor worked before they saw it. So far he has sold over 300,000 copies of Turbo Pascal, which is making it the standard of Pascals.

Currently, about 50,000 people know MPE and 20,000,000 peole know MS-DOS. To display a file to the screen, MPE requires

```
FCOPY FROM=filename;TO
```

MS-DOS requires

```
TYPE filename
```

Worse, to send a file to the printer, MPE requires

```
FILE T;DEV=LP
FCOPY FROM=filename;TO=*T
```

MS-DOS requires

```
PRINT filename
```

And to copy a file to a new file, MPE requires

```
FCOPY FROM=oldfile;TO=newfile;NEW
```

MS-DOS requires

```
COPY oldfile newfile
```

On the :DBEXPRESS distribution tape we have created UDC's that look like the MS-DOS commands on the theory that people will appreciate having to type fewer key strokes to get the same result. We also expect that many of the users will have had access to Vectras or other MS-DOS PC's in their college or work experience. Type, Print and Copy have been in the public domain for years. And there is no copyright law that says you can't simplify work.

It appears as though we were anticipating a move by HP. MPE-XL will also have some of these simplified commands.

Conclusion

The world of microcomputers is already user-sympathetic. The reason is that it is a very competitive environment, populated not by captive slaves, but users who have limited time and limited budgets.

For the last few years, in the HP 3000 world, we have been able to ignore that competition for users' hearts and minds because our minicomputers were their only option. That is no longer true and user hostile and programmer competent are no longer acceptable. We must offer axles for the users and not wheels from the programmers.

-30-

Million Record Database Strategies

by

Cliff Lazar

SystemsExpress

15015 Ventura Blvd.
Sherman Oaks, CA 91403
818/784-6966

SystemsExpress

Million Record Database Strategies

Table of Contents

Introduction
- Evolutionary Performance Degradation
- Database Mental Retardation

Mice are Different from Elephants
Dr. Ruth's Good and Bad Databases
Access Strategies
- HP Manual Way
- Programmer Intensive
- Partitioned Datasets
- The Entity Detail
  - Blocking Factors

Backup Strategies
Archive Strategies -- the WORM Drive
- Read-only Datasets
- Dormant On-site Compressed Data
- Dormant Off-line, On-site Compressed Data

Conclusions

Introduction

Million record databases are the justification for many of the HP sites in our community. Million record databases are also the source of many of the problems for those HP sites.

For our purposes, a million record database can be comprised of a few datasets with over 100,000 records each or a massive dataset with over a million records. Such a single massive set is unlikely because it would tend to fall of its own weight without support from side index files.

If there are any who will volunteer that they have such a massive stand-alone file, we should all have sympathy for them.

Some large databases start off strutting with good performanace and then evolve into stumbling poor performers. Others, because of poor design, start off poorly and just get worse. In this article I will look at some of the causes of poor database performance on the megabase level and suggest some strategies for improving performance.

In general, the strategies that work for million record, seven figure databases should also apply to six figure and to a degree, even to high four figure databases. Three figure databases, with capacities under 1000, live within the power of the computer, and poor design is compensated by expensive hardware.

Evolutionary Performance Degradation

In some cases the bases grow from modest beginnings and as a result grow to become inefficient and counter-productive in an evolutionary manner. An observation of psychologists is that if a stimulus changes less than 2% of the base amount, most humans will not be able to tell the difference. Another observation is that if system performances are jumping up and down around a trend, most humans will not be able to discern the trend without keeping records and computing running averages.

Chronic gamblers suffer from fixating on episodic reinforcement; chronic optimists never forget the times the system gives them three second response time and overlook average five minute waits as aberrations. Serial searches in hashed masters will, by the nature of the hash, episodically give very quick responses.

Thus, when a database, with a poor design, begins its life with little data and few users, its performance is very good. Most retrievals occur at finger speed and batch jobs consume very little CPU and even wall clock time. As the base grows and the capacities are reset to higher figures things begin to slow down a little bit at a time.

If the transaction rates are relatively constant, such as sales orders or work hours, the actual percentage increases in database

size will decrease each period. Thus, when things begin to suffer the percentage increases in response time may be below the threshold of perception.

Parallel with creeping degradation, typically, the staffing of the data entry and reporting functions are increasing and being trained, so there is little room for comparison based on extended observation.

Database Mental Retardation

When a baby is born the general expectation is that it will develop into a normally intelligent human with reasonable response time. During the infancy we don't expect the baby to act like a normal adult.

During childhood we watch for signs of improvement. During adolecence we tolerate accidents and lapses because we love our offspring but at some point, usually during childhood, if we don't see improvement and a trend toward normal intelligence we may be forced to admit that our child has chronic problem that won't be solved with age.

Some databases are that way. Their physical structure is such that they will never have normally intelligent response times. Databases with capacities under 1000 entries, while having chronic physical problems, don't exhibit them because their size can be overwhelmed by the computer's speed.

Million record databases with chronic physical problems will be irretrievably inefficient. Adding a disk drive won't help. Going from one MIP to four MIPS won't help. Cacheing will contribute little or no relief. The design is not only inefficient, it is big and inefficient.

Mice are Different than Elephants

Mice and Elephants are both mammals, with common characteristics like vertibrae, hair, breasts and well-developed brains. At the same time they are very different because of their size. In some ways the elephants suffer from diseconomies of scale. Feeding them becomes a major logistical problem. Elephants can practically deforest an area as they feed. They endanger other nearby animals. Cleaning up after them is a signifcant issue if you own some. They require alot of real estate, exponentially more, not arithmetically more than mice.

As with mega databases, Elephants require greater strength in their structure. Their muscles are a higher percentage of slow twitch--they can't react rapidly. Their nervous systems require greater decentralization because the signals have a greater distance to travel. They require specially trained handlers.

You can have hundreds of mice in your barn yard and not notice them, but one elephant will make a big impression.

Million record databases have similar diseconomies of scale. The obvious issues are

- o Floor space for the disk drives
- o Response time for retrievals
- o Backup time
- o Integrity of the data
- o Dedicated skill levels needed for maintenance

HP and other manufacturer's are offering a solution to the floor space problem with the 4 gigabytes that will squeeze into the same six square feet as an HP7933 disk drive. This still ignores, or even contributes to back-up and archive problems.

Responsiveness or Dr. Ruth's Good and Bad Databases

A good database is one that will allow the users to retrieve their data when they want it.

Conversely a bad database will not yield data as fast as the user wants and often not as fast as a manual alternative.

The goodness and badness of a database is a function of its size more than its structure. In the case of million record databases this becomes very apparent.

Consider a box of 10 driver's licenses. You can spread them out on a table and get whatever information you want instantly. You don't need Image. You don't even need a computer! But what if you have 10,000 driver's licenses in the same box. Now you can't find anything you want.

How will you sort your driver's licenses? By the meaningless driver's license number (which is the equivalent of the Manual Master Unique key), by last name, by city, by sex? Any sort will obviate any other sort, if only one sort is possible as in the case of an Image manual master. What is the solution?

For decades, libraries have maintained multiple copies of their cards: by author, subject and title. You can go into a library with a million books and find the reference card you want in less than two minutes because you do a a random access search on a sorted key and not a serial search.

A million record master on an HP 3000/68 will take 5.8 hours for the average stand-alone serial search. In a multi-user environment every other process will suffer, and the search will take longer.

If the dataset only has fifty to a hundred customer records then retrieval takes place at finger speed and its O.K. to have a manual master with serial search.

So let's draw an inference. Small masters are O.K. Large mas-

ters, with over 1000 entries, make bad databases. A million record manual master makes an abomination.

Next time someone wants to sell you a package, don't accept a demonstration based on less than 100,000 records, if that is going to be your environment.

Anyone who buys a database system which will become 400 byte wide masters with 100,000 to 1,000,000+ capacity should have his head examined. **You ignore diseconomies of scale at your peril.** You should always get a performance guarantee for the likely range of set capacities your shop will use.

Physical Realities

A million record database has certain physical realities that are inherant in its size. At 400 bytes wide, a million records will contain 400 million bytes. If the data is stored in a master then 1.25 million records will be needed to avoid the migrating secondaries. That means that the disk drives will need 500 million bytes. Until now that required two 7933 drives and at least one GIC.

Access Strategies

Four Possible Approaches

The are four possible approaches to the design of a customer information file:

- The HP Manual Way - put the customer data in a master file
- The programmer intensive Way --Keep the information in the master but build a set of key index files to compensate for the unsearchability of the master. Many programmers have homegrown approaches to this and Omnidex offers a canned IMSAM approach for about $10,000.
- Partitioned Datasets -- The data are partitioned into groupings that vary from high probability of hits to very low probability. The high probability set is searched first. If it is small enough it may reside in cache memory most of the time and the responses may be at CPU speed.
- The Entity Detail -- store the customer information in a detail but assure that the entries are unique. Systems-Express, our company, offers a retrofit system to copy the data from masters to details and provide rapid search with a combination of automatic masters and a KSAM pointer file, if generic search is desired. The cost varies from $3,200 to $6,000, if Cobol source is desired. The approach can be used to redesign an existing database or make easy-access extract databases for

inquiries and adhoc reports.

The HP Manual Way - put the customer data in a master file

I feel I have to appologize to the HP community. Early on, I wrote a beginner's guide to building Image databases. It was based on the HP Image manual and my readings in the various proceedings. The basic approach was that entities should be stored in Image masters and transactions should be stored in details.

The problem arises when the user wants to extract data not based on the meaningless unique key of the master but on some remem-berable item such as a company name, or a contact name, or a city, or even a phone number.

I have spoken to numerous users who have bought or built megabases with 300,000 record masters with no pointers except the meaningless key. The key must be meaningless or it couldn't be unique, if there are over 100,000 of them. Exception: part numbers with some rational naming convention. These users tell me that if they want non-key retrievals they must set up stream jobs to run over night or over the week-end.

Other users tell me that they make their living just building duplicative extract files so that the users can get the data they need when they need it.

Let me recant: Don't build databases with the entities in the manual masters. The only thing that Masters are good for is spell-checking.

Put entity data in details and point to them with automatic masters or KSAM files or MPE side files or IMSAM files. More on details below.

Programmer Intensive--Keep the information in the master, but build pointer files

The operational word is "keep." The assumption is that you inherited the database from someone who was fired or was promoted before management understood the problem.

For technical or political reasons you can't change the structure of the database and convert the masters to details. The solution to getting improved performance is to create side index files which point to the key in the master. KSAM, MPE and IMSAM are good approaches. The appeal of KSAM is that it comes free with HP 3000's and you don't have to build search software.

The approach we follow is fill the KSAM file with the correct spelling of the key values, and not record numbers that are subject to rapid change. This will help to avoid maintaining the KSAM file every time a record is deleted.

Partitioned Datasets -- The data are partitioned into groupings that vary from high probability of hits to very low probability

Consider a transaction detail with one million records, which describe events that take place over time. While it may be advantageous to keep all the records online, it may be possible partition the detail into a few datasets, one or two of which have high probabilities of search hits:

- o Current Day's input
- o Latest Week
- o Latest Month
- o Residual of the data

The Current Day's and Latest Week may have the highest probability of search hits and also will be the smaller of the files. With large blocking factors and a big cache much of the high probability files could be RAM resident most of the time.

At the end of each of the data collection periods the data is posted to the next more comprehensive dataset and deleted from its original source.

The smaller files also have the advantage that if there is a system failure, less is involved in reconstructing the smaller files.

The Entity Detail -- Store the customer information in a detail but assure that the entries are unique

A entity detail is a detail with one or more keys that are pointed to with automatic masters where at least one key is limited to a chain length of one. The result is that the key is unique, just like a master. The advantage is that there can be as many as 16 keys and that the detail itself does not require excess space for migrating secondaries. The automatic masters do.

Typically a detail may have three to seven keys and typically only one will be unique.

We add a KSAM pointer file for all the keys that we would like generic search. Generally only one to three of the keys will be designated as generic search keys. The KSAM file will contain as many records as the combined capacities of the automatic masters and the manual masters that are to be pointed to, and the record size is equal to the largest field to be pointed to. Thus, this side index file makes trading space for time a very significant tradeoff. At the same time many of our users appreciate the ease of retrieval that generic search offers to the online user.

The typical response time for a large database with a KSAM front-end generic search is less than one second. There is no noticeable difference for a direct key search, with the full

value spelling, bypassing the KSAM look-up.

In Table 1 are displayed the theoretical space requirements and search times for a standard manual master and an entity detail. The Image implicit pointer bytes have been ignored. The search time estimates include the time to serial search through empty records in the master. The searches for data in the details assumes that keys are available in automatic masters.

Table 1 - Theoretical Space Requirements for 1,000,000 Customer Records and Search Times .
(MM=000,000)
(400 byte wide record with 100 bytes of keys)

Structure	Keys	Bytes in Masters (1,2)	Bytes Detail	Bytes in KSAM File (3)	Total Bytes (4)	Search Time (5)
Standard All Manual	1	500MM	0	0	500MM	5.8 hours
Detail with 5 5 automatics	5	125MM	400MM	0	525MM	1 second
Detail with 5 5 automatics 1 KSAM with 5 pointers	5	125MM	400MM	125MM	650MM	<1 second

Footnotes:
(1) 400 bytes * 1MM * 1.25 = 500MM for migrating secondaries avoidance
(2) automatic masters = 100 bytes * 1MM * 1.25 = 125MM
Some of the automatics may be much less than 1MM
(3) 20 bytes * 5 * 1MM * 1.25 = 125MM
(4) Sum of Bytes in Masters, details and KSAM files
(5) 500MM/30 accesses per second/3600 seconds per hour/2
This can be substantially inproved through better blocking factors
< 1 second adjusts for the savings in user keystrokes with generic search

Entity details are used by some shops as extract databases for quick user access to the data that is otherwise locked in ugly masters. :DBTRANS or Supertool is used to copy the data and :DBEXPRESS or grunt Cobol is used to build the retrieval system.

Blocking Factors

Well selected blocking factors can substantially improve the efficiency of database retrievals. The more records that are retrieved with each seek, the less time and resouce consuming seeks are necessary.

The common wisdom has been that 25% ofa master's space should be empty space to avoid migrating secondaries outside of the block. It has been suggested (Van Valkenburgh, Interex 86, Detroit Paper 3105, p.23) that the free space can be limited to the reciprocal

of the blocking factor. I have run a simple simulation of this approach and it appears that for high blocking factors, the likelyhood of not finding free space in any given block is high enough to expect a lot of migrating secondaries to cross over the block boundaries if the free space is limited to the reciprocal.

This is another good reason to avoid using masters except for spell checkers.

Backup Strategies

Million record databases require lots of backup. While backing up only changed datasets is attractive there may be a problem with partial database backups that can jepordize database integrity. It may be that keeping the high probability smaller datasets in a separate database can allow nightly backup of only the smaller sets. This can save substantial operator time.

Archive Strategies -- the WORM Drive

Currently archiving million byte databases poses a dual problem of database security and timely archive access.

Security considerations require that backups be kept off site to prevent catastrophic loss and unintentional operator destruction of what look like convenient scratch tapes. Million record databases require large tape storage. Now is the time to consider adopting the write once read many (WORM) optical storage.

WORM drives, shown at COMDEX have capacities in the multiple trillions of bytes. They can deliver any record within 30 seconds and occupy eight square feet of floor space.

Two copies of archival data can be made, with one stored off-site and the other stored conveniently on-site and possibly even on-line in an archival database. The archival database need be opened only when the older data was required.

Implementation of WORM technology will change current database practices in many HP shops. Much of the data currently kept on disks has little current access, but is stored in the database because archiving is so inconvenient if the data is unexpectedly needed. It is another example of eposodic reinforcement. The one time in two years when old data was needed, it took hours or days to get it back on the machine so the Database Administrator decided to keep as much data on-line as there was floor space available.

Duplicate WORM optical storage provides the dual advantages of security from loss and accidental scratch over-writes and rapid access to historic data. Experience suggests that the access programs, along with documentation, be stored with the data. It is likely that the programs will change over time and current programs will be unable to read the old data.

Hewlett-Packard is currently examining the WORM technology and may have a product on the market in the near future. Other vendors have WORM systems available now that can be interfaced to the HP 3000.

Read-only Datasets

An intermediate solution to the current non-existance of the WORM and the desire to keep megabytes of data online but with smaller disk impact is the concept of a "Read-only Dataset."

A read-only dataset is compressed into 25% or less space that an uncompressed dataset but can only be read from and not written to. Much of the data that we keep online in read/write datasets is, in fact, read-only data. It is historical--employee work hours or customer payments that we want to read but also don't want to overt overwrite.

One college in California keeps all its student records online since 1929. They are never written to, only read from, after the close of the school year. Those records occupy eight 7933H disk drives, about four gigabytes. It's all read-only data.

There are other examples of read-only data:

- Technical Abstacts
- Property Records
- Library Book References
- Stock Transactions
- Chemical Formulas and Mathematical Equations
- Census Data
- Voting Patterns
- Telemetry/Sensor Readings

This class of data can be called compressed but read-only.

Dormant On-line Compressed Data

It is also possible to compress data and store it on-line in a dormant state and then decompress it when it is needed. This has the advantage that it is available within minutes but it is not readable or writeable. Such data can be kept online while the image of it is archived at a remote location. The cost is relatively low to have this option. My company offers such a compression/decompression tool. It's called :COMPFILE. You can have your data both ways--archived and available but out of the way.

Dormant Off-line, On-site Compressed Data

The offline conterpart of WORM is highly compressed data on tape. The advantage is that you can save tape, save backup time and you can keep potentially need copies of the data onsite. We of course offer :COMPTAPE for this function.

Conclusions

Up to the current time many million record databases have suffered from diseconomies of scale. They are so large and so poorly designed that they perform poorly. Maintaining their security and integrity has resulted in a bureaucratic environment where response time has suffered and improvements were avoided.

Technology available now can allow the users of million record databases to gain more rapid access to the data through use of entity details or pointer files or extract files. The response time improvements for unkeyed versus keyed access is thousands of seconds versus sub-second response.

Switching from a master to a detail with automatic masters achieves the sub-second response time at the cost of increasing the database size by about 5% to 25%. Adding a generic search KSAM can double the data space requirements, but makes access more user-sympathetic.

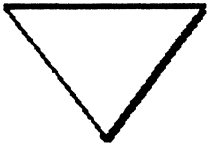
Shifting from tape archiving to WORM optical can increase data security while improving data accessability.

Exploiting data compression techniques promises to give the user site more rapid access to data, with increased security while achieving substantial savings in disk space.

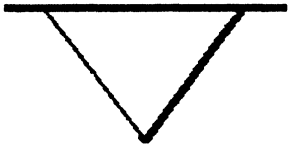
While the majority of the databases are small and reasonably efficient, my observations indicate that the majority of the disk space in the HP community is occupied by large, inefficient databases that can be improved with reasonable effort.

SystemsExpress

Old and New Symbols



Manual
Master



Automatic
Master

Detail

Compressed
Master

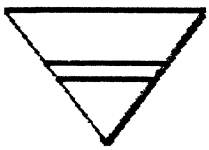
Compressed
Detail

SystemsExpress

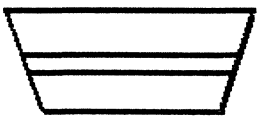
Extraction Symbols

Extractor

Selective
Extractor



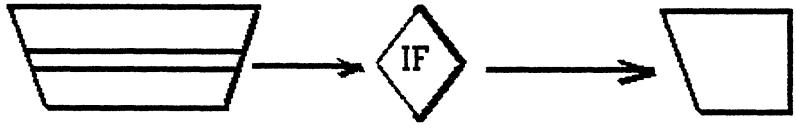
Extractee
Master



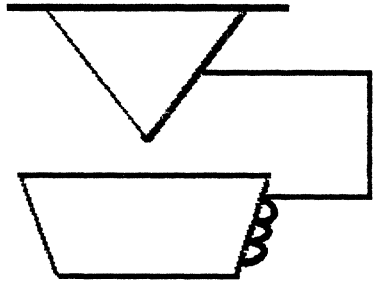
Extractee
Detail

SystemsExpress

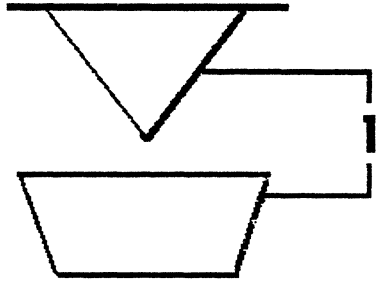
Extractions



Chains



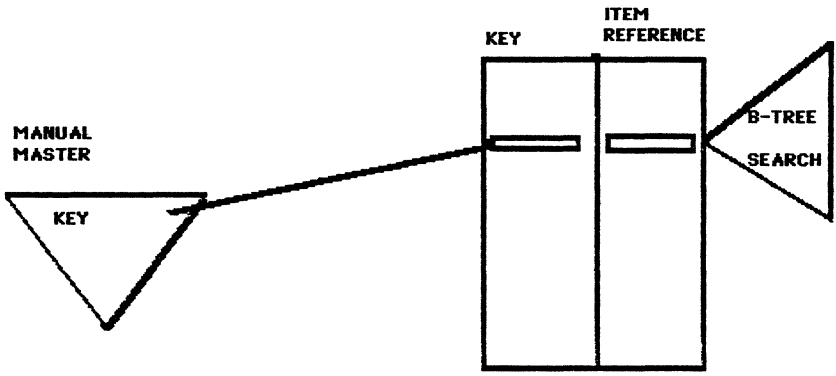
Normal Detail
Normal Chain



Entity Detail
Chain length = 1

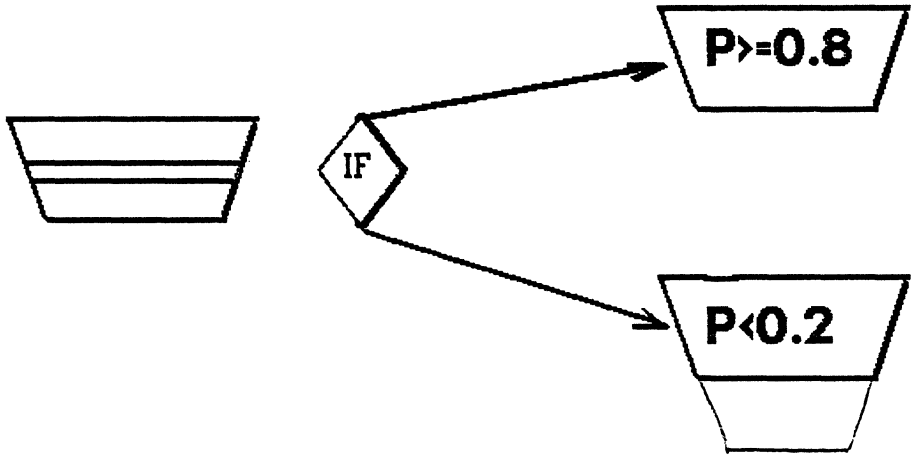
Programmer Intensive

-pointer files



SystemsExpress

Partitioned Datasets



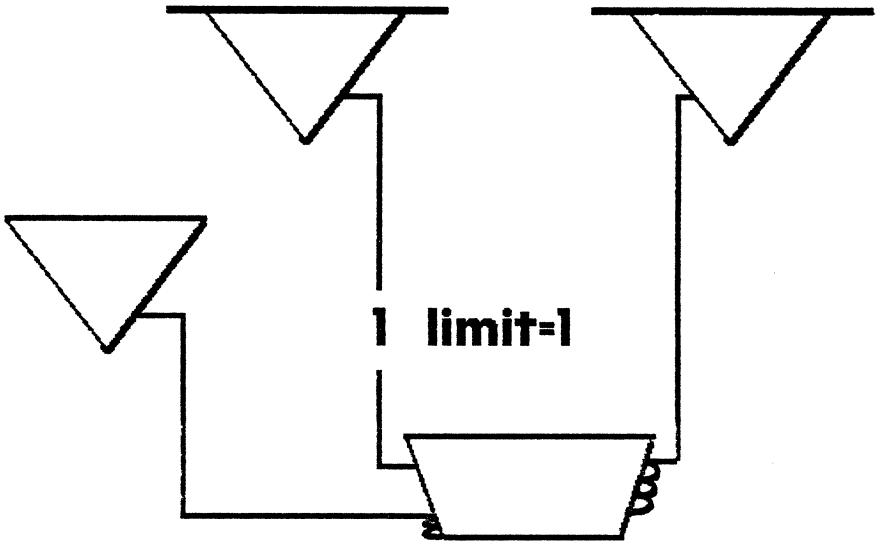
Must be:

Mutually exclusive and
Collectively exhaustive

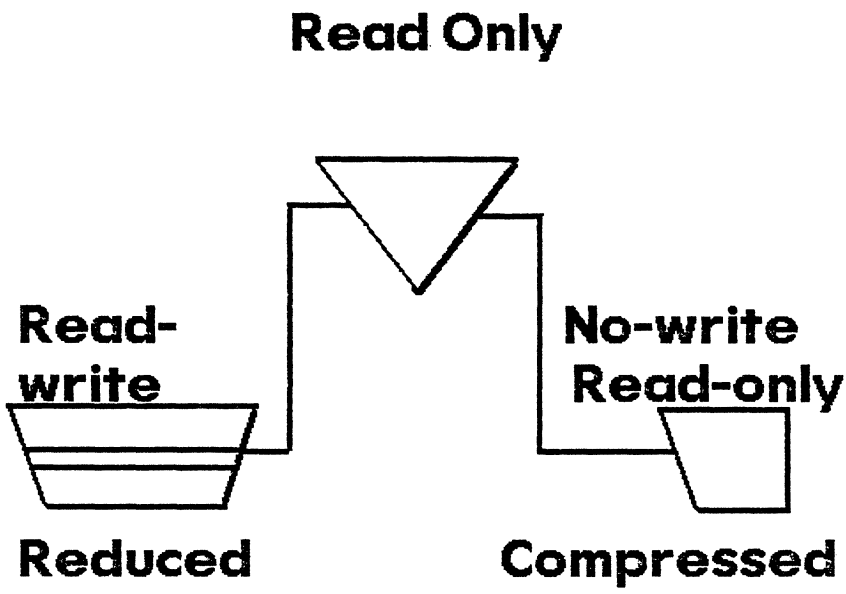
$$\sum P=1.0$$

SystemsExpress

Entity Detail



SystemsExpress



KSAM ISSUES AND UTILIZATION

Timothy G. Leadbetter

1. OVERVIEW

The HP 3000 Computer System's Fundamental Operating System includes various data management tools. For purposes of this discussion, data management tools will be defined as features of the system software which facilitate collection and storage of data.

The primary tool covered will be KSAM, including storage formats and methods As a comparison, IMAGE Database Management software will be discussed in overview. These and the sequential, or "flat", file comprise the data storage methods available within the Fundamental Operating System of the HP 3000 Computer System.

The discussion will cover basic data structures used by KSAM and IMAGE, then provide an overview of these tools. KSAM strengths and weaknesses will be outlined, and KSAM-specific file system errors explained. Potential uses for KSAM will be presented, as well as situations where it is best avoided. An explanation of the basis for these recommendations will also be given. Design considerations and tradeoffs will be presented, along with tools to predict disc space use and forecast the number of levels in your key trees. Recovery from file corruption and system interrupts will be presented, as well as corrective actions for KSAM related file system errors.

2. DATA STRUCTURES

The primary data structures used in KSAM are the calculated random sequential file and the balanced tree. IMAGE uses hashed random access files and linked lists for storage and access of data. To provide a background for comparisons, these storage methods will be defined. These definitions will be fairly brief, with additional readings listed for those who would like a more detailed description.

2.1 LINKED LISTS

A linked list is a physically linear structure with it's nodes linked along logical lines. These links are accomplished through pointers, which define chains. A linked list may be singly or doubly linked. The singly linked list will contain data and a pointer to the next element in the chain. A doubly linked list also contains a backward pointer, allowing the chain to be read first to last, last to first, and which permits reversing directions if needed. Each linked list will contain a head entry, which contains pointers to the first and, optionally, last data entry. A chain count may be included as a check for integrity. IMAGE uses linked lists, with chain counts, to manage paths, synonym chains (hashing collisions), and free lists.

The fundamental drawback to linked lists is the potential for broken chains. These may occur due to software aborts, hardware failures, or procedural errors. Repair of a broken chain is not a trivial matter, and assistance should be sought if any uncertainty exists.

A general rule as to the number of looks it will take to locate an entry is difficult to put forward here. This is a function of the method used to place the chain head for linked list. On average over time, half of the chain length will be traversed to find an entry within the chain. The number of looks to find the chain head will be added to this to determine the total number of records examined to reach a desired entry.

2.2 HASHED RANDOM ACCESS FILES

In a random access file, the data is arranged in linear records, but any record can be instantly accessed (without serially reading it's predecessors) by providing a relative record number or word offset into the file. The hashing feature provides a random element to insertion, and provides rapid access to an entry

with a specific value. The degree of randomization is a function of the hashing algorithm used and the capacity of the file. In general, the sparser the population, the less likely collisions are to occur.

The principal problems seen with hashed direct access files are collisions, where more than one entry hash to the same location, and space requirements. In order to reduce population density, the file will need to have some minimum amount of free space, usually expressed as a percentage of the total space in the file. With large capacities, this can become quite large. A common way of handling collisions is to create a synonym chain, using the linked list from the preceding paragraph to contain the secondaries. The physical placement of the secondaries varies, but IMAGE searches forward to find the next open record, then links this record into the synonym chain.

In general, the number of looks required to locate an entry in a hashed direct access file will be 1 plus half the length of the average synonym chain. The length of the synonym chain is effected by the capacity of the file, the density of the population, and the hashing algorithm used. With many algorithms the capacity of the file is a factor in the calculation, and use of prime numbers for the capacity will reduce collisions, speeding access.

2.3 TREES

Trees are a specialized form of graph, where each level contains entries and pointers to further entries in the next level. The entries in a node are in order (ascending or descending) by value within the node, and values not found in this node but existent in the tree will be located in another level. Nodes are defined as "root", "branch", and "leaf", where the root is the first level encountered in accessing the tree, a branch is a level between the root and leaves, and a leaf is the final level encountered. A tree will contain exactly one root, but may contain many branches and leaves. The arrangement and number of records and pointers in a node is a function of design.

In a balanced tree, an equilibrium is sought between nodes. Insertion and deletion algorithms vary, but generally follow these rules:

INSERTION:

A. The record is inserted into the correct node based on the sequence specified, and in the correct location. If the node is at capacity, it is extended temporarily to permit balancing.

B. If the node was full prior to addition of the new record, it is split. and the value at the value at the center of the node is inserted into the next higher level. This is applied to higher levels as needed.

C. If the root node is full, and the split proceeds to that level, the root node is split, with the center entry becoming the sole entry in the new new root node, and a new level formed immediately beneath it.

DELETION:

A. The specified record is removed from it's node.

B. If the node is now at less than half of it's capacity, it is contracted into the adjacent nodes.

2.4 SEQUENTIAL FILES

In a sequential file, records are added by appending to the current end of file, without regard for content. This works well for fairly static data, but introduces inefficiency in the process of locating a desired record. On average, it is necessary to look at half of the records to locate the one being sought. In a large file, this adds considerable overhead.

KSAM addresses this inefficiency by using a calculated read into the sequential file. This means that the relative record number is provided, and the file system directly reads that record, resulting in a single look. The factor which adds overhead is the structure used to provide the relative record number. This type of access is referred to as calculated, directed, or random.

2.5 SUMMARY OF DATA STRUCTURES

In this section, we have discussed the strengths and weaknesses of four data storage schemes in common use. This information will be used as background for the material on KSAM, and will, hopefully, make the workings of KSAM's files and operation more understandable.

3 KSAM COMPONENTS

KSAM files consist of two MPE files, which the file system treats as a single entity for KSAM access. This means that if either of them is corrupted or destroyed, recovery will be required before access can be restored. In this section, we will talk about KSAM key and data files and their interaction, and will look at the overhead involved in adding and deleting a record.

3.1 KEY FILE

The KSAM key file contains overhead information, followed by a key tree for each tree defined. The overhead information is a 128 word record, which contains information on the file and access. The name of the data file is stored here, as well as a time stamp. The number of records in the data file is recorded here, as well as a counter for the number of times each intrinsic has been called against the file. A complete breakdown of this area has been excerpted from the KSAM/3000 Reference Manual and is included in the Appendix.

The next structure in the key file is the key descriptor block, which is 128 words long and contains the definitions of your keys. The layout of this block has also been included in the Appendix.

The final information in the key file are the key entry blocks for each key. These are kept in B-tree representation.

The key file does not contain a KSAM user label. It's filecode is 1080, and is reported as KSAMK in LISTF output.

3.2 DATA FILE

The data file is essentially a sequential file, with the addition of a user label to relate it to the key file. This label contains the name of the key file, and nothing more. It is not accessible by the user. The data file has a filecode of 0, but LISTF reports the code to be KSAM. Records are appended to the end of the file, and an entry is made in each key tree in the key file, allowing calculated access to the data file in KSAM reads by key value.

3.3 ADDING AND DELETING RECORDS

It is the addition and deletion of records that are costly with a KSAM file. In the best case, the data record must be written, then it's corresponding key entry placed. This will involve locating the adjacent values in each key tree and inserting or deleting the entry. In the case of a deletion, the data record must first be located and read prior to being rewritten, as it is flagged as deleted, rather than being made available for reuse.

In the worst case, the key insertion or deletion may cause the number of levels in one or more B-tree to change, which is not a trivial matter. We will go over a brief example, to illustrate the effect of dynamic data on a KSAM file.

In KSAM, key values are stored in ascending order. Each entry is bounded by pointers. The pointer to the left of the value points to a block in the next lower level where all values are lower than the this one. The pointer to the right points to a block in the next lower level where all values are higher than the current entry. In KSAM, both pointers lead to the same number of levels. Thus, the tree is said to be balanced.

When a key value is requested, a binary search is performed on the root block. If the key value is found, the search ends. If the value is not in the root, the appropriate pointer is used to progress to the next block, which also has a binary search performed. These steps repeat until the value is found or the indicated leaf node is searched without finding the value. Using this method, the approximate average number of records examined to find an existing entry will be LOG2(n), where n is the capacity of the file.

Before discussing addition of key values, an examination of the basic B-tree is in order. Figure 1 illustrates a simple tree. Note that the root node contains only one entry, but the branches are all at least half full. In this example, the key blocking factor is four. In practice, there are usually considerably more entries per block.

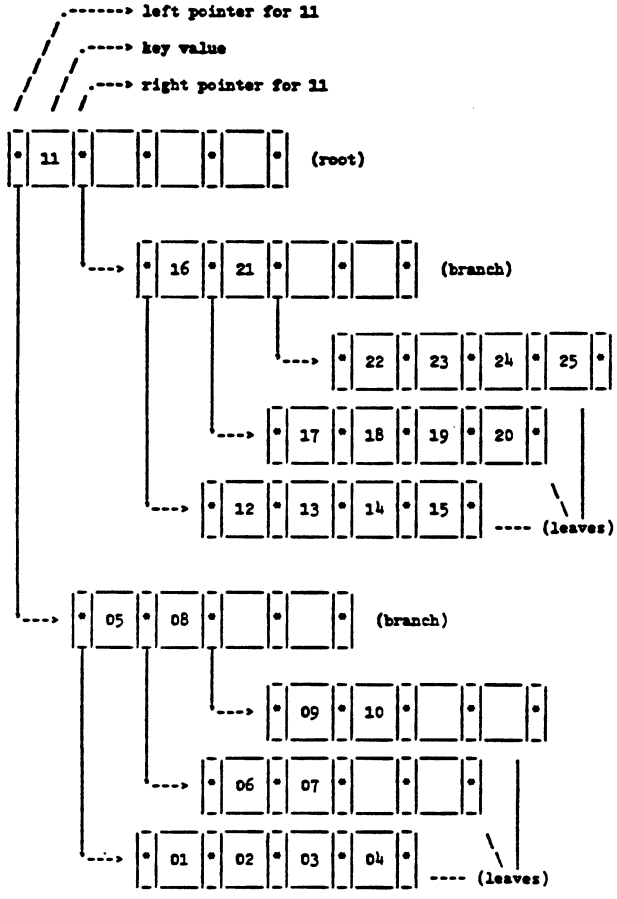


Figure 1
Sample Key Tree

3.3.1 KEY ADDITION

When a key value is added to a B-tree, it is inserted into a leaf node. If the appropriate leaf is full, a block split will occur. In KSAM, a block split is done by inserting the new value into it's proper position, then migrating the middle value of the block to the next higher level and dividing the remaining entries into two blocks, based on the side of the previous middle entry on which they resided. Each of the resulting blocks will be half full.

This insertion method is complicated by the possibility of duplicate keys. When you build your KSAM file, you can specify whether or not duplicates are allowed in each key. If allowed, you can force them to be added in chronological order or allow the duplicates to be randomly inserted. If they are randomly inserted, KSAM will insert the new key entry into an appropriate block which avoids block splits, if possible.

Figures 2a, 2b, and 2c illustrate the development of the tree as entries are inserted. In the initial block, we see that there is room for one more entry in the root node. A value is added, which fills the root node but requires no further action.

The next addition forces the root block to split. This results in a root node with one entry and pointers to two leaves, each of which is half full.

The next sequence of additions fills one leaf, forcing it to split. Note that the split adds an entry to the root node and results in a total of three leaves, but that no intermediate branch level is formed. The next two addition sequences also add an entry to the root, add leaves, but do not add branches.

The final set of additions again causes a leaf node to split, but the root node is full. This requires that the root block split, and a branch level is formed.

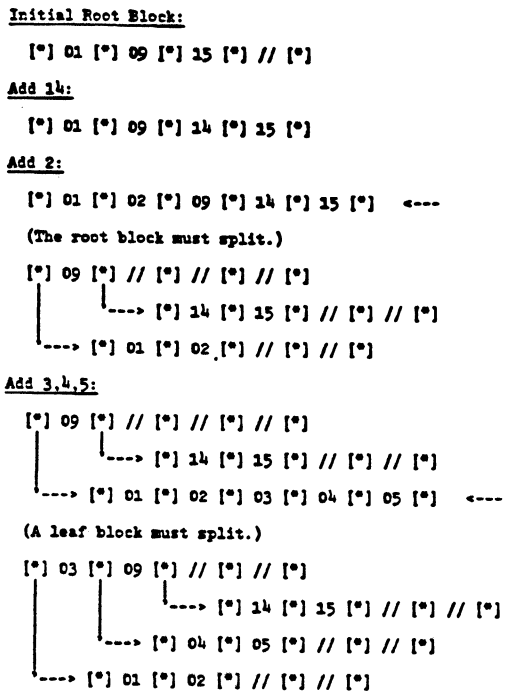


Figure 2a
Insertion of Key Entries (1 of 3)

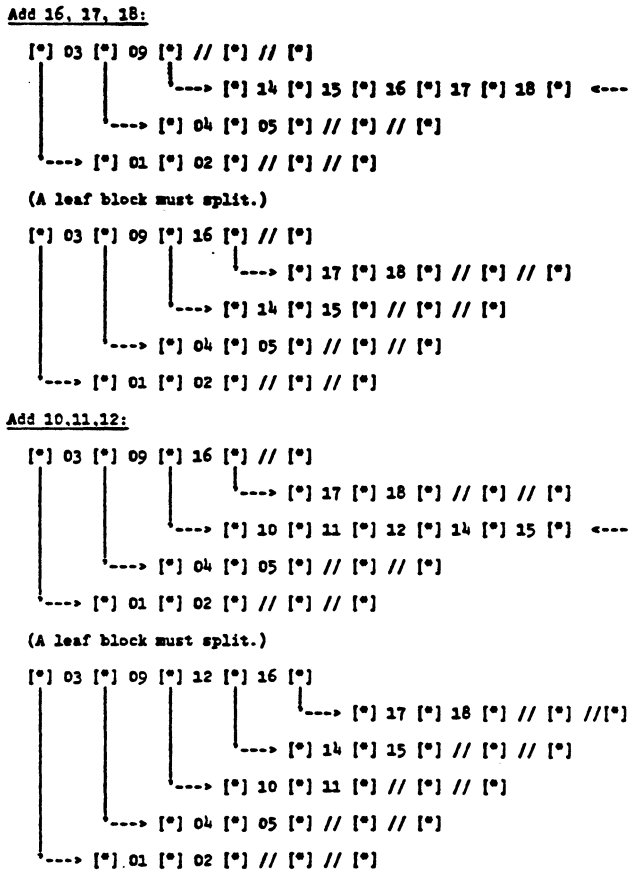


Figure 2b
Insertion of Key Entries (2 of 3)

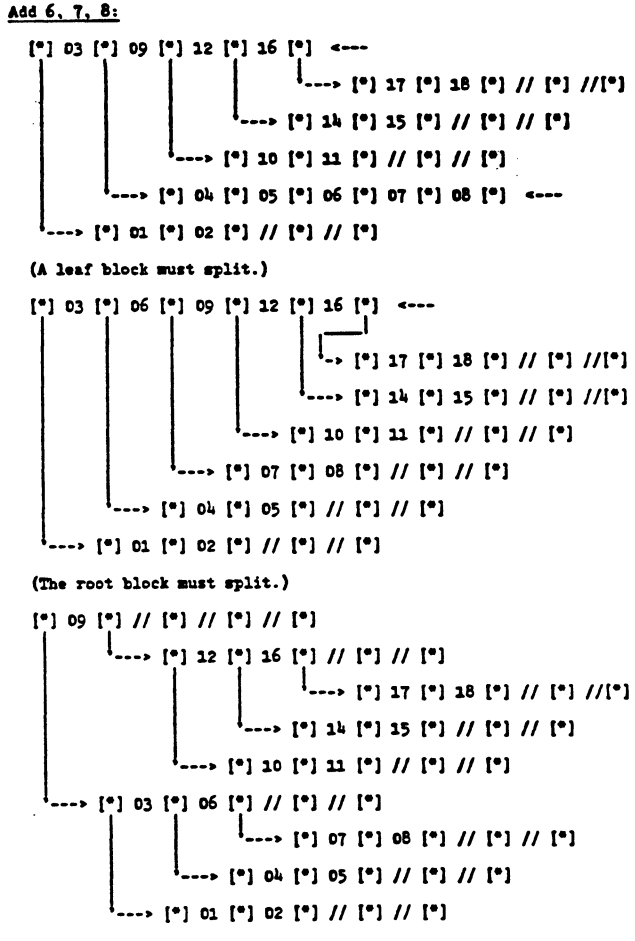


Figure 2c
Insertion of Key Entries (3 of 3)

3.3.2 KEY DELETION

Deletion of an entry is subject to the restriction that a leaf must be at least half full. If deletion of a key value causes it's block to become less than half full, the block must be balanced or merged.

Balancing is accomplished by taking values from adjacent leaf blocks that are more than half full. If no adjacent leaf block is less than half full, the block deleted from is merged with an adjacent block. When this is done, the value which separates the blocks at the next higher level is moved into the new block, and the adjacent blocks are joined with it to form a block.

Merging may cause the block at the next higher level to become less than half full, so merging may ripple up the tree. This continues until all blocks but the root block are at least half full.

Figures 3a through 3d contain an example of deletion of key values. The initial tree contains a single value in the root block. The first deletion removes this value, so the next value in the tree moves up to take it's place. This results in one leaf being less than half full, so it is merged with the adjacent leaf. This results in the branch being less than half full, so the branches are balanced.

The next deletion is fairly straightforward. The value being deleted resides in a branch, and results in the branch being less than half full. The branch is balanced with it's adjacent branch, which is over half full.

The third deletion causes a leaf to become less than half full. Again, the leaf is balanced with an adjacent leaf.

The final deletion is again of the sole value in the root node. Moving the next value up to replace it causes a leaf to become less than half full. As no adjacent leaf is more than half full, two leaves must be merged. Doing so causes a branch to become less than half full, and it's adjacent branch is not more than half full, so the branches are merged. The result is a full root block, pointing to leaves without intermediate branches.

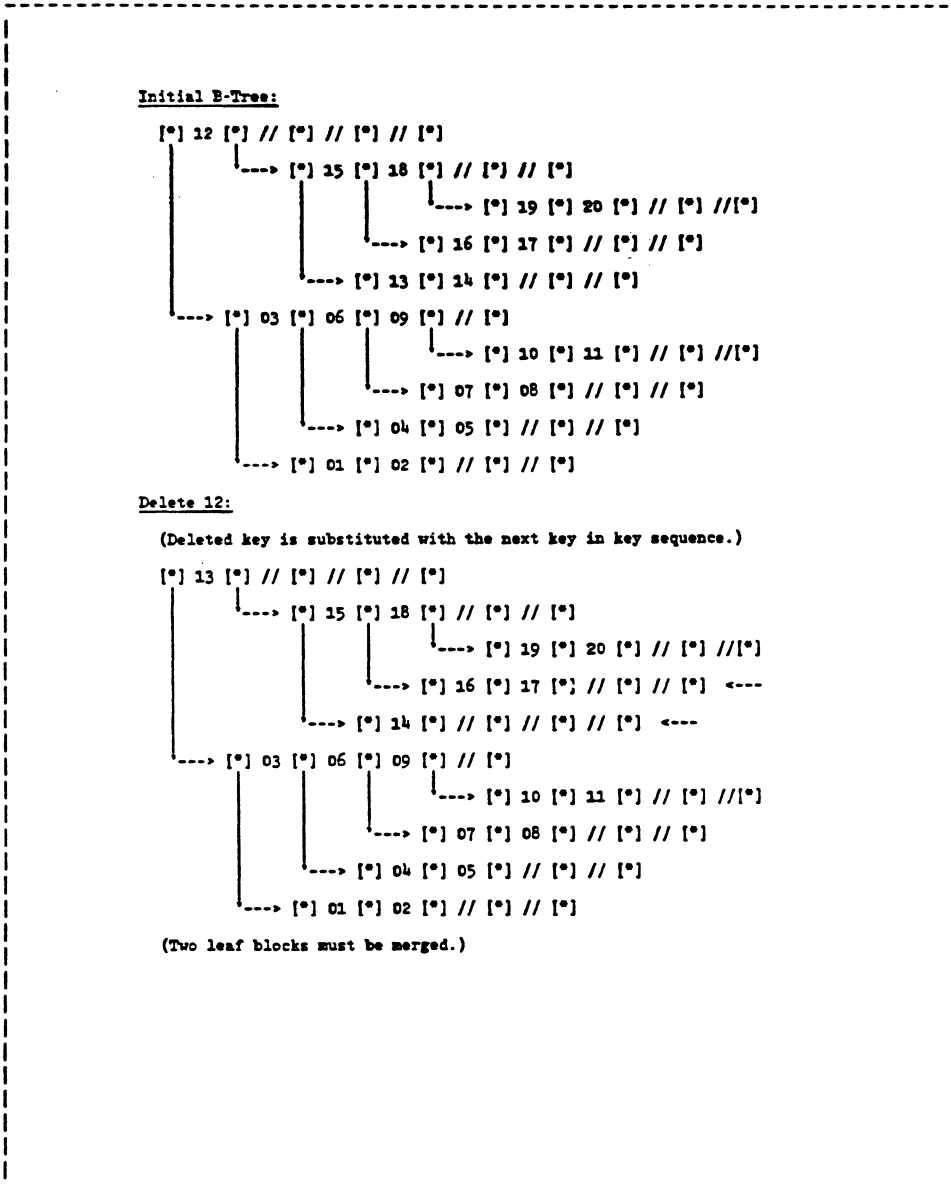
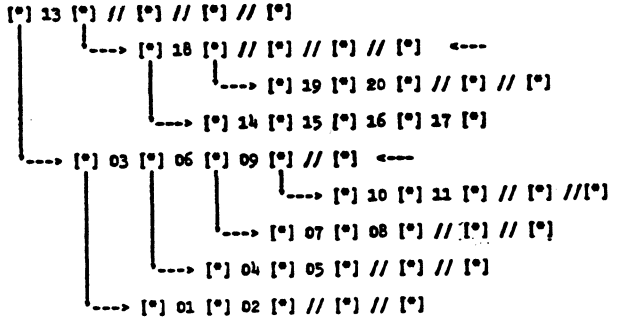


Figure 3a
Deletion of Key Entries (1 of 4)



(Two branch blocks must be balanced.)

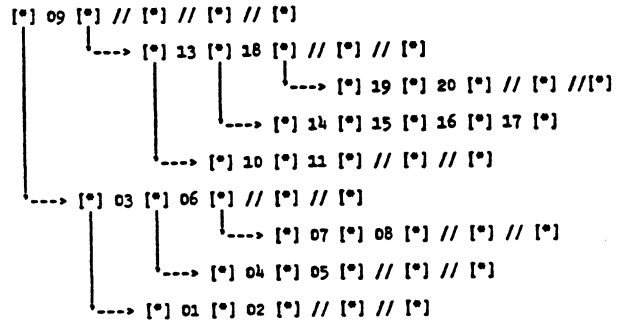


Figure 3b
Deletion of Key Entries (2 of 4)

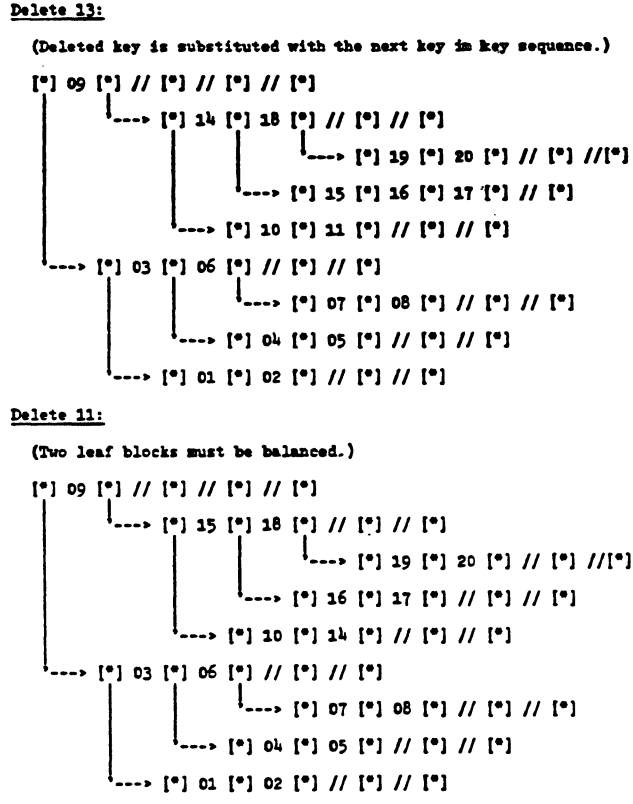
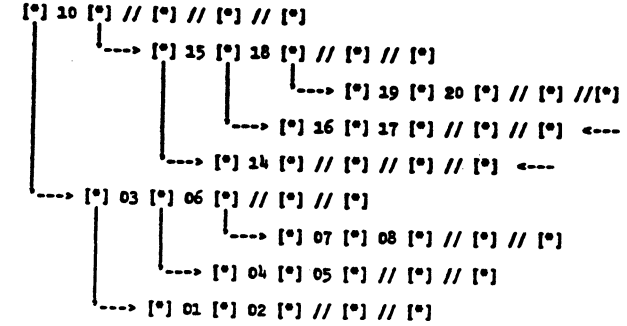


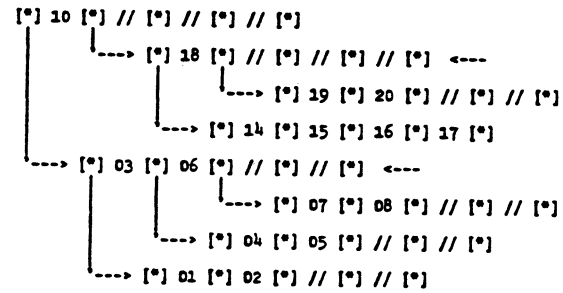
Figure 3c
Deletion of Key Entries (3 of 4)

Delete 09:

(Deleted key is substituted with the next key in key sequence.)



(Two leaf blocks must be merged.)



(Two branch blocks must be merged.)

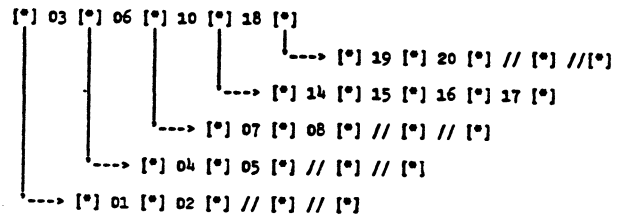


Figure 3d
Deletion of Key Entries (4 of 4)

3.4 KSAM STRENGTHS AND WEAKNESSES

The preceding paragraph pretty well covers the down side of KSAM. In a dynamic environment, tree splits and contractions will cause slow updates. For this reason, updates are best done as batch processes, to minimize the inconvenience for your user community.

On the positive side, KSAM will allow partial key lookup, is relatively fast in accessing a desired record, and can be structured to be reasonable in disc space consumption. KSAM files are easily defined, easily loaded, and easily maintained.

In contrast, IMAGE excels in an interactive update environment, provides good integrity for physical data, and can be structured with greater flexibility. On the negative side, it is more difficult to define a database than a KSAM file, requires programming effort to load, and can be difficult to repair in the event of damage to one of the datasets.

3.5 APPLICATIONS FOR KSAM . . . AND PLACES TO AVOID IT

KSAM provides a unique function in the form of partial key lockups. An example can be seen in part numbers, where the next higher assembly provides a prefix to the component's part number. If the next higher assembly is A123B, and we want all components with "6" as the first digit, we can supply the key value "A123B6" and retrieve all part numbers beginning with that combination. Other examples are telephone area codes or prefixes, zip codes, employee numbers, and so on.

KSAM also provides rapid conversion from sequential access files, particularly where the data will remain fairly static. The conversion is done by using KSAMUTIL to build the new file, then using FCOPY to copy the data and build the key trees. In this regard, it is also useful where an application must be developed rapidly, uses data of limited complexity, and will predominantly perform lockups. An example of this is an index, such as a part number file, vendor file, employee file, and so forth.

If your application will require access to data with complex interactions, or where the data will be dynamic, IMAGE excels. KSAM has the drawbacks of many pairs of files to represent complex data, where redundancy could be reduced by use of a database. There are also the added drawbacks of tree splits and contractions on addition and deletion.

Access to IMAGE master datasets will generally take less time than accesses to a KSAM file. In the case of a well designed IMAGE database, this advantage is very pronounced. But KSAM carries advantages in development and maintenance time, and is used successfully by many HP 3000 users. It provides easy migration for the new HP 3000 user, with good compatibility with ISAM files. The tools to develop and maintain KSAM files are quick to learn and use, and recovery from most problems is straightforward.

Specific places to use KSAM will be where partial key lookup are needed, where data is fairly static, where development time is constrained, or where converting from another keyed or indexed system. KSAM will also be useful where data is normally stored in a sequential file, but a rapid access inquiry program is needed.

KSAM should be avoided where the data is very dynamic. The overhead of key file management should be avoided. If you must use KSAM to provide partial key inquiries, try to structure updates to be performed in batch mode as much as possible. KSAM is also inefficient for storage of a large set of related data. Use of IMAGE will reduce data redundancy and reduce concurrent access headaches.

4. RECOVERY OF KSAM FILES

Most problems with KSAM files are correctable using KSAMUTIL or FCOPY. The common problems can be broadly grouped as user error induced or system failure induced.

User errors are typically due to purging or renaming either the data or key file with the MPE commands. The PURGE and RENAME commands don't check for KSAM, so they only operate on the specified member and leave the other. This produces some unexpected errors. Your program may report File System Error 52 (non-existent permanent file) for a file that shows up in a LISTF. You may also receive a File System Error 100 (duplicate permanent file name) if you attempt to build a KSAM file in your program or with KSAMUTIL. In either case, you need to determine whether the key or data file is missing. If the key file is missing, recovery is fairly simple, and the risk of data loss is low.

You may also encounter a File System Error which is specific to KSAM. Common examples are FSERR 171 (duplicate key value, duplicates not allowed), FSERR 173 (target count larger than record size), FSERR 175 (key value for a record to be deleted is not in the key file), and various others. The generally result from a system failure, where KSAM was interrupted in the the process of updating the files. A file in this condition generally starts by reporting FSERR 192 (file system error occurred while the KSAM file was open). The following sections will cover repairing the most common errors and aborts. As a first step, do a LISTF on your data and key files. If one is missing, refer to the appropriate paragraph. Most other errors will be corrected by using FCOPY to rebuild the key structure.

4.1 MISSING KEY FILE

If your LISTF shows that the data file exists but the key file is missing, or it's name is not the same as it was built with, the following actions will build a new keyfile in the absence of the original.

```
:RUN KSAMUTIL.PUB.SYS
>BUILD newdata; . . .    match the build for the old file, using
                         a different keyfile name
>EXIT
:FCOPY FROM=olddata;TO=newdata;NOKSAM;NOUSERLABELS
:RENAME olddata,archivename
:RUN KSAMUTIL.PUB.SYS
>RENAME newdata,olddata
>RENAME newkey,oldkey
>EXIT
```

These steps built the new key and data file, then loaded them from the orphaned data file. NOKSAM is specified to make the file system ignore the absence of the old key file. If omitted, you'll receive FSERR 52. The next step is to "archive" the old data as a safeguard. If you have some problem in the KSAMUTIL RENAMEs, you can use the old data file to redo this sequence. The last step is to rename the new data and key files to match the old ones. This must be done in KSAMUTIL, and the files must be specified separately.

4.2 MISSING DATA FILE

If you find that the data file is missing, the steps to recover are more likely to result in data loss. Your insurance here is in a backup schedule which minimizes the opportunity for lost data. Recovery is accomplished by restoring the old data and key files, followed by reapplying transactions done between the backup and loss of the file. Since the key file contains only the key fields from the data file, the data file cannot be rebuilt with only the keyfile as a source.

A scheme to minimize the loss would be to maintain a parallel MPE sequential file, containing all records being added. This simple approach will not cover loss of updates or deletions, but will at least track

additions. You can implement a more complex derivative using user logging or more complete records in the sequential file. Check with your local HP office for help with developing a recovery plan for your important files.

4.3 SYSTEM FAILURE WHILE KSAM FILES WERE OPEN

This problem is reported via FSERR 192, and is recovered from by running KSAMUTIL and issuing ">KEYINFO filename;RECOVER" against all KSAM files.

4.4 INCONSISTENCIES BETWEEN KEY AND DATA FILES

In several situations it is possible to have the data file updated but not the keyfile. This may show up as FSERR 171 or FSERR 175, as well as others. This FCOPY will correct most errors not fixed in the previous procedures, with minimum data loss. First, we'll look at the command sequence, then we'll discuss the steps.

```
:FCOPY FROM=olddata;TO=(newdata,newkey);NEW;KEY=0
:RUN KSAMUTIL
>PURGE olddata
>RENAME newdata,olddata
>RENAME newkey,oldkey
>EXIT
```

The FCOPY creates a new KSAM file (key and data), using the names you provided within parentheses. The parentheses must be included in the FCOPY command. The ";KEY=0" clause tells FCOPY that the source file is a KSAM file, but that it's key structure is not to be copied. This will insure that the key file matches the data in the new data file. The next step is to use KSAMUTIL to purge (or rename) the old file. If you purge, you need only give the data file name. Rename requires that both key and data be renamed. The final step is to rename the new key and data files to the old names, so your programs can access them.

The ";KEY=n" clause can also be used to force the file to be loaded based on a secondary key. This is

useful where an application mix has changed, and a secondary key is used more frequently than the primary key.

5. FILE DESIGN CONSIDERATIONS

The parameters used to build a KSAM file will impact it's efficiency in terms of disc utilization and access time. As all key blocks are initially allocated when the file is built, the key file will typically be quite large. Along with the full initial allocation, this size is due to the control block and key descriptors, and the need to maintain pointers for tree levels, as well as relative record numbers in the data file for each key value. The size factor increases with the number of keys. Large key blocking factors also result in increased disc usage. This is due to the fact that key blocks tend to be between half full and full. As the capacity of the block increases, the potential for empty entries increases.

On the other hand, decreasing key blocking factors will increase insertion and deletion time due to more frequent need for block size adjustment, and will increase the number of levels to be searched, which will require additional disc I/O operations.

Design of a KSAM file is a tradeoff between these factors. The tools presented here are intended to assist you in evaluating the space requirements, and in determining the appropriate parameters to be supplied to build a file.

We will first consider the data file, as it is the most straightforward. The key file will then be considered. The keyfile, as mentioned above, has the greatest potential for wasted space, and the most influence on access to the KSAM file, so it's parameters must be given reasonable consideration.

5.1 DATA FILE

The most frequently encountered problem with the data file is using the default blocking factor. For sequential files, MPE's file system calculates a blocking factor which is reasonably efficient in use of disc space. KSAMUTIL will generally provide the least efficient space utilization if the default is taken, as the default blocking factor is always one. To illuminate this, if you build a KSAM file with six byte data records, the default blocking factor will place one record in a block. The minimum block size is 256 bytes, so 250 bytes are lost in each block. For a file with 1024 records, you are losing 256,000 bytes of disc space.

To minimize this waste, a blocking factor should be chosen where the record size in bytes multiplied by

the blocking factor yields a number as close to an even multiple of 256 as possible, without exceeding the multiple of 256. You may want to write a program to do this for you. Consider having the program request a minimum acceptable utilization and maximum acceptable blocking factor, and listing those factors which fit the specified criteria. If the resultant blocksize is at or over 32767 bytes, the build will fail. If it is between 16383 and 32767, any later access must specify only one buffer when the file is opened. If this is not done, an error will result on access, due to inability to fit the buffers into a single extra data segment.

The data file work sheet in Appendix B is part of an overall set of worksheets to evaluate space requirements. The instruction sheet explains the use of each section, and the steps to complete it. Notice the two columns, headed FIXED and VARIABLE. These refer to the record type in your file. If you use variable length records, additional overhead is incurred.

5.2 KEY FILE

The last two pages in Appendix B apply to the key file. Again, each section is explained in the instructions which precede the worksheets. You will only use the second page if your file contains more than four keys, and will need to make further pages if you have more than eight keys.

Key blocking factors can be significant in access performance and in disc space utilization. Remember that an inverse relationship exists - larger key blocks speed access but require more disc space. KSAM defaults to a key blocking factor which yields blocks as close as possible to 1024 words in length. This may not provide the best mix in your application. The tables in Appendix A are used to determine an initial key blocking factor. This will be refined by the calculations in the worksheet, and may change slightly.

Table A-1 will give you the key block size based on key blocking factor and record size. The column in the center shows the default for various key sizes. Once you have determined the key blocking factor to be used, this table will approximate the size of disc transfers to read the key trees.

Table A-2 is used to determine the approximate number of tree levels you will encounter for each key blocking factor. It can be used to determine the key blocking factor based on performance needs. Keep the number of levels at or below three for applications with high performance requirements. This will use more disc space than smaller blocks and more levels would, but will reduce disc reads and block splits.

Determine the maximum number of levels your application can tolerate, then look down that column to find a key blocking factor which fits. As an example, lets assume that you can tolerate three levels in your tree, and will have up to 1000 records in your file. Let's also assume that this file has one key of 40 bytes. We look down column three and find a critical value just greater than 1000 - in this case it is with a key blocking factor of 14. Referring again to table A-1, we find that a KBF of 14 and key length of 40 bytes, the block will require three sectors.

6. FILE SYSTEM ERRORS

There are a few file system errors which are commonly seen with KSAM files, and a few more that are only seen with KSAM files. This section provides a short description of each, and corrective actions to be taken.

FSERR 52 (nonexistent permanent file). This is commonly seen when either the key or data file has been renamed or purged outside of KSAMUTIL. If a rename is done, the relationship between the key and data file is lost, since the key file control block and data file user label are not updated. The action to remedy this was discussed earlier, and will depend on whether the key or data file is missing.

FSERR 74 (not enough room in stack for another file). This occurs when there is not enough room in the file area of your stack to open another file. This may also appear with IMAGE datasets or MPE files, but KSAM opens two files and has more opportunity to cause this. Actions are to close all unused files prior to opening this one or reduce the MAXDATA with which the program was PREPPED or run.

FSERR 100 (duplicate permanent file name). This may arise through using MPE to purge a KSAM file, rather than using KSAMUTIL. This leaves the file not specified (generally the key file) on the system, and the error is reported if the file is rebuilt. The corrective action is to use MPE to purge the remaining file.

FSERR 107 (insufficient space for user labels). This arises when FCOPYing an existing KSAM file into another KSAM file. The corrective action is to specify ;NOUSERLABELS in the FCOPY, to avoid bringing the old file's user label to the new file.

FSERR 171 (duplicate key value). This occurs when a system interrupt occurs during posting of a deletion. The data record is flagged for deletion, but the key entry remains. This will carry into new files unless ;KEY=0 is specified in the FCOPY.

7. INCREASING THE SIZE OF A KSAM FILE

One of the common administrative actions taken is to increase file size. With KSAM, this is relatively simple. You will issue a file equation for a new data file, specifying only the name and number of records, then use FCOPY to build the new key and data file, as follows.

```
:FILE newdata;REC=numrecs
:FCOPY FROM=olddata;TO=(newdata,newkey);NEW;NOUSERLABELS
:RUN KSAMUTIL.PUB.SYS
>PURGE olddata
>RENAME newdata,olddata
>RENAME newkey,oldkey
>EXIT
```

This will build and load the new file with two MPE commands, then use KSAMUTIL to rename the files so the applications can access them.

8. SUMMARY

In this paper, we have discussed common KSAM issues. The underlying data structures have been touched upon, and their application within KSAM explored. The common operations of adding and deleting records have been reviewed with attention to the overhead involved.

KSAM strengths and weaknesses have been related to potential applications. We have also looked at resolution of common problems with KSAM files.

File design considerations were discussed, and specific tools provided to determine parameters to be used in building KSAM files. The tradeoffs of disc utilization and access time were related to these parameters. Common file system errors were covered, and a quick method to increase KSAM file capacity was presented

The information and tools presented here should help you determine where to use KSAM, and to use it effectively where it is appropriate.

BIBLIOGRAPHY

"KSAM/3000 Reference Manual", Hewlett Packard Company part number 30000-90079

Knuth, Donald E.: "The Art of Computer Programming, vol. 1, Fundamental Algorithms", second edition, Addison-Wesley Publishing Company, Reading, Mass., 1973

Sorenson, P. G and J. P. Tremblay: "An Introduction to Data Structures with Applications", McGraw-Hill Book Company, New York, N.Y., 1976

K S A M Table A-1

KSAM KEY BLOCKING FACTOR FOR KEY LENGTH VS BLOCK SIZE

KBF \ KEY BLOCK SIZE IN SECTORS	1	2	3	4	5	6	7	1K: 8:	9	10	11	12	13	14	15	2K 16
2	24	50	74	100	126	152	178	:202:	228	254	280	306	330	356	382	408
K 4	20	40	62	84	104	126	148	:168:	190	212	232	254	276	296	318	340
E 6	16	34	54	72	90	108	126	:144:	162	182	200	218	236	254	272	290
Y 8	14	30	46	62	78	94	110	:126:	142	158	174	190	206	222	238	254
10	12	26	42	56	70	84	98	:112:	126	140	154	170	184	198	212	226
L 12	12	24	36	50	62	76	88	:100:	114	126	140	152	164	178	190	204
E 14	10	22	34	46	56	68	80	: 92:	104	114	126	138	150	162	174	184
N 16	10	20	30	42	52	62	74	: 84:	94	106	116	126	138	148	158	170
G 18	8	18	28	38	48	58	68	: 78:	88	98	106	116	126	136	146	156
T 20	8	16	26	36	44	54	62	: 72:	80	90	100	108	118	126	136	144
H 22	8	16	24	32	42	50	58	: 66:	76	84	92	102	110	118	126	136
24	6	14	22	30	38	46	54	: 62:	70	78	86	94	102	110	118	126
I 26	6	14	22	28	36	44	52	: 58:	66	74	82	90	96	104	112	120
N 28	6	12	20	28	34	42	48	: 56:	62	70	76	84	92	98	106	112
30	6	12	18	26	32	40	46	: 52:	60	66	72	80	86	94	100	106
B 32	6	12	18	24	30	38	44	: 50:	56	62	70	76	82	88	94	102
Y 34	4	10	18	24	30	36	42	: 48:	54	60	66	72	78	84	90	96
T 36	4	10	16	22	28	34	40	: 46:	52	56	62	68	74	80	86	92
E 38	4	10	16	22	26	32	38	: 44:	48	54	60	66	72	76	82	88
S 40	4	10	14	20	26	30	36	: 42:	46	52	58	62	68	74	78	84
42	4	10	14	20	24	30	34	: 40:	44	50	56	60	66	70	76	80
44	4	8	14	18	24	28	34	: 38:	44	48	52	58	62	68	72	78
46	4	8	14	18	22	28	32	: 36:	42	46	50	56	60	66	70	74
48	4	8	12	18	22	26	30	: 36:	40	44	50	54	58	62	68	72
50	4	8	12	16	20	26	30	: 34:	38	42	48	52	56	60	66	70

^ default value

NOTE:
The key length is only shown in even number of bytes since the key entries are allocated on word boundaries.

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level 1	2	3	4	5
	1	5	17	53	161
K 4	4	16	52	160	484
	4	24	124	624	3,124
E	1	7	31	127	511
6	6	30	126	510	2,046
Y	6	48	342	2,400	16,806
	1	9	49	249	1,249
8	8	48	248	1,248	6,248
	8	80	728	6,560	59,048
B	1	11	71	431	2,591
10	10	70	430	2,590	15,550
L	10	120	1,330	14,640	161,050
	1	13	97	685	4,801
O 12	12	96	684	4,800	33,612
	12	168	2,196	28,560	371,292
C	1	15	127	1,023	8,191
14	14	126	1,022	8,190	65,534
K	14	224	3,374	50,624	759,374
	1	17	161	1,457	13,121
16	16	160	1,456	13,120	118,096
	16	288	4,912	83,520	1,419,856
F	1	19	199	1,999	19,999
18	18	198	1,998	19,998	199,998
A	18	360	6,858	130,320	2,476,098
	1	21	241	2,661	29,281
C 20	20	240	2,660	29,280	322,100
	20	440	9,260	194,480	4,084,100
T	1	23	287	3,455	41,471
22	22	286	3,454	41,470	497,662
O	22	528	12,166	279,840	6,436,342
	1	25	337	4,393	57,121
R 24	24	336	4,392	57,120	742,584
	24	624	15,624	390,624	9,765,624
	1	27	391	5,487	76,831
26	26	390	5,486	76,830	1,075,646
	26	728	19,682	531,440	14,348,906
	1	29	449	6,749	101,249
28	28	448	6,748	101,248	1,518,748
	28	840	24,388	707,280	20,511,148
	1	31	511	8,191	131,071
30	30	510	8,190	131,070	2,097,150
	30	960	29,790	923,520	28,629,150
	1	33	577	9,825	167,041
32	32	576	9,824	167,040	2,839,712
	32	1,088	35,936	1,185,920	39,135,392
	1	35	647	11,663	209,951
34	34	646	11,662	209,950	3,779,134
	34	1,224	42,874	1,500,624	52,521,874

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min / Crit / Max	B-tree level 1	2	3	4	5
	1	37	721	13,717	260,641
K 36	36	720	13,716	260,640	4,952,196
___	36	1,368	50,652	1,874,160	69,343,956
E	1	39	799	15,999	319,999
38	38	798	15,998	319,998	6,399,998
Y ___	38	1,520	59,318	2,313,440	90,224,198
	1	41	881	18,521	388,961
40	40	880	18,520	388,960	8,168,200
___	40	1,680	68,920	2,825,760	115,856,200
B	1	43	967	21,295	468,511
42	42	966	21,294	468,510	10,307,262
L ___	42	1,848	79,506	3,418,800	147,008,442
	1	45	1,057	24,333	559,681
O 44	44	1,056	24,332	559,680	12,872,684
___	44	2,024	91,124	4,100,624	184,528,124
C	1	47	1,151	27,647	663,551
46	46	1,150	27,646	663,550	15,925,246
K ___	46	2,208	103,822	4,879,680	229,345,006
	1	49	1,249	31,249	781,249
48	48	1,248	31,248	781,248	19,531,248
___	48	2,400	117,648	5,764,800	282,475,248
F	1	51	1,351	35,151	913,951
50	50	1,350	35,150	913,950	23,762,750
A ___	50	2,600	132,650	6,765,200	345,025,250
	1	53	1,457	39,365	1,062,881
C 52	52	1,456	39,364	1,062,880	28,697,812
___	52	2,808	148,876	7,890,480	418,195,492
T	1	55	1,567	43,903	1,229,311
54	54	1,566	43,902	1,229,310	34,420,734
O ___	54	3,024	166,374	9,150,624	503,284,374
	1	57	1,681	48,777	1,414,561
R 56	56	1,680	48,776	1,414,560	41,022,296
___	56	3,248	185,192	10,556,000	601,692,056
	1	59	1,799	53,999	1,619,999
58	58	1,798	53,998	1,619,998	48,599,998
___	58	3,480	205,378	12,117,360	714,924,298
	1	61	1,921	59,581	1,847,041
60	60	1,920	59,580	1,847,040	57,258,300
___	60	3,720	226,980	13,845,840	844,596,300
	1	63	2,047	65,535	2,097,151
62	62	2,046	65,534	2,097,150	67,108,862
___	62	3,968	250,046	15,752,960	992,436,542
	1	67	2,311	78,607	2,672,671
66	66	2,310	78,606	2,672,670	90,870,846
___	66	4,488	300,762	20,151,120	1,350,125,106
	1	69	2,449	85,749	3,001,249
68	68	2,448	85,748	3,001,248	105,043,748
___	68	4,760	328,508	22,667,120	1,564,031,348

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level 1	2	3	4	5
	1	71	2,591	93,311	3,359,231
K 70	70	2,590	93,310	3,359,230	120,932,350
	70	5,040	357,910	25,411,680	1,804,229,350
E	1	73	2,737	101,305	3,748,321
72	72	2,736	101,304	3,748,320	138,687,912
Y	72	5,328	389,016	28,398,240	2,073,071,592
	1	75	2,887	109,743	4,170,271
74	74	2,886	109,742	4,170,270	158,470,334
	74	5,624	421,874	31,640,624	##############
B	1	77	3,041	118,637	4,626,881
76	76	3,040	118,636	4,626,880	180,448,396
L	76	5,928	456,532	35,153,040	##############
	1	79	3,199	127,999	5,119,999
O 78	78	3,198	127,998	5,119,998	204,799,998
	78	6,240	493,038	38,950,080	##############
C	1	81	3,361	137,841	5,651,521
80	80	3,360	137,840	5,651,520	231,712,400
K	80	6,560	531,440	43,046,720	##############
	1	83	3,527	148,175	6,223,391
82	82	3,526	148,174	6,223,390	261,382,462
	82	6,888	571,786	47,458,320	##############
F	1	85	3,697	159,013	6,837,601
84	84	3,696	159,012	6,837,600	294,016,884
A	84	7,224	614,124	52,200,624	##############
	1	87	3,871	170,367	7,496,191
C 86	86	3,870	170,366	7,496,190	329,832,446
	86	7,568	658,502	57,289,760	##############
T	1	89	4,049	182,249	8,201,249
88	88	4,048	182,248	8,201,248	369,056,248
O	88	7,920	704,968	62,742,240	##############
	1	91	4,231	194,671	8,954,911
R 90	90	4,230	194,670	8,954,910	411,925,950
	90	8,280	753,570	68,574,960	##############
	1	93	4,417	207,645	9,759,361
92	92	4,416	207,644	9,759,360	458,690,012
	92	8,648	804,356	74,805,200	##############
	1	95	4,607	221,183	10,616,831
94	94	4,606	221,182	10,616,830	509,607,934
	94	9,024	857,374	81,450,624	##############
	1	97	4,801	235,297	11,529,601
96	96	4,800	235,296	11,529,600	564,950,496
	96	9,408	912,672	88,529,280	##############
	1	99	4,999	249,999	12,499,999
98	98	4,998	249,998	12,499,998	624,999,998
	98	9,800	970,298	96,059,600	##############
	1	101	5,201	265,301	13,530,401
100	100	5,200	265,300	13,530,400	690,050,500
	100	10,200	1,030,300	104,060,400	##############

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level				
	1	2	3	4	5
	1	103	5,407	281,215	14,623,231
K 102	102	5,406	281,214	14,623,230	760,408,062
	102	10,608	1,092,726	112,550,880	##############
E	1	105	5,617	297,753	15,780,961
104	104	5,616	297,752	15,780,960	836,390,984
Y	104	11,024	1,157,624	121,550,624	##############
	1	107	5,831	314,927	17,006,111
106	106	5,830	314,926	17,006,110	918,330,046
	106	11,448	1,225,042	131,079,600	##############
B	1	109	6,049	332,749	18,301,249
108	108	6,048	332,748	18,301,248	1,006,568,748
L	108	11,880	1,295,028	141,158,160	##############
	1	111	6,271	351,231	19,668,991
O 110	110	6,270	351,230	19,668,990	1,101,463,550
	110	12,320	1,367,630	151,807,040	##############
C	1	113	6,497	370,385	21,112,001
112	112	6,496	370,384	21,112,000	1,203,384,112
K	112	12,768	1,442,896	163,047,360	##############
	1	115	6,727	390,223	22,632,991
114	114	6,726	390,222	22,632,990	1,312,713,534
	114	13,224	1,520,874	174,900,624	##############
F	1	117	6,961	410,757	24,234,721
116	116	6,960	410,756	24,234,720	1,429,848,596
A	116	13,688	1,601,612	187,388,720	##############
	1	119	7,199	431,999	25,919,999
C 118	118	7,198	431,998	25,919,998	1,555,199,998
	118	14,160	1,685,158	200,533,920	##############
T	1	121	7,441	453,961	27,691,681
120	120	7,440	453,960	27,691,680	1,689,192,600
O	120	14,640	1,771,560	214,358,880	##############
	1	127	8,191	524,287	33,554,431
R 126	126	8,190	524,286	33,554,430	2,147,483,646
	126	16,128	2,048,382	260,144,640	##############
	1	137	9,521	657,017	45,334,241
136	136	9,520	657,016	45,334,240	##############
	136	18,768	2,571,352	352,275,360	##############
	1	139	9,799	685,999	48,019,999
138	138	9,798	685,998	48,019,998	##############
	138	19,320	2,685,618	373,301,040	##############
	1	141	10,081	715,821	50,823,361
140	140	10,080	715,820	50,823,360	##############
	140	19,880	2,803,220	395,254,160	##############
	1	143	10,367	746,495	53,747,711
142	142	10,366	746,494	53,747,710	##############
	142	20,448	2,924,206	418,161,600	##############
	1	145	10,657	778,033	56,796,481
144	144	10,656	778,032	56,796,480	##############
	144	21,024	3,048,624	442,050,624	##############

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level 1	2	3	4	5
	1	147	10,951	810,447	59,973,151
K 146	146	10,950	810,446	59,973,150	##############
	146	21,608	3,176,522	466,948,880	##############
E	1	149	11,249	843,749	63,281,249
148	148	11,248	843,748	63,281,248	##############
Y	148	22,200	3,307,948	492,884,400	##############
	1	151	11,551	877,951	66,724,351
150	150	11,550	877,950	66,724,350	##############
	150	22,800	3,442,950	519,885,600	##############
B	1	153	11,857	913,065	70,306,081
152	152	11,856	913,064	70,306,080	##############
L	152	23,408	3,581,576	547,981,280	##############
	1	155	12,167	949,103	74,030,111
O 154	154	12,166	949,102	74,030,110	##############
	154	24,024	3,723,874	577,200,624	##############
C	1	157	12,481	986,077	77,900,161
156	156	12,480	986,076	77,900,160	##############
K	156	24,648	3,869,892	607,573,200	##############
	1	159	12,799	1,023,999	81,919,999
158	158	12,798	1,023,998	81,919,998	##############
	158	25,280	4,019,678	639,128,960	##############
F	1	163	13,447	1,102,735	90,424,351
162	162	13,446	1,102,734	90,424,350	##############
A	162	26,568	4,330,746	705,911,760	##############
	1	165	13,777	1,143,573	94,916,641
C 164	164	13,776	1,143,572	94,916,640	##############
	164	27,224	4,492,124	741,200,624	##############
T	1	169	14,449	1,228,249	104,401,249
168	168	14,448	1,228,248	104,401,248	##############
O	168	28,560	4,826,808	815,730,720	##############
	1	171	14,791	1,272,111	109,401,631
R 170	170	14,790	1,272,110	109,401,630	##############
	170	29,240	5,000,210	855,036,080	##############
	1	175	15,487	1,362,943	119,939,071
174	174	15,486	1,362,942	119,939,070	##############
	174	30,624	5,359,374	937,890,624	##############
	1	179	16,199	1,457,999	131,219,999
178	178	16,198	1,457,998	131,219,998	##############
	178	32,040	5,735,338	1,026,625,680	##############
	1	183	16,927	1,557,375	143,278,591
182	182	16,926	1,557,374	143,278,590	##############
	182	33,488	6,128,486	1,121,513,120	##############
	1	185	17,297	1,608,713	149,610,401
184	184	17,296	1,608,712	149,610,400	##############
	184	34,224	6,331,624	1,171,350,624	##############
	1	191	18,431	1,769,471	169,869,311
190	190	18,430	1,769,470	169,869,310	##############
	190	36,480	6,967,870	1,330,863,360	##############

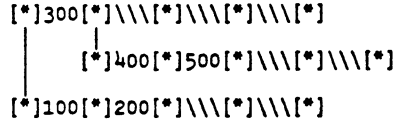
TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level 1	2	3	4	5
	1	199	19,999	1,999,999	199,999,999
K 198	198	19,998	1,999,998	199,999,998	##############
	198	39,600	7,880,598	1,568,239,200	##############
E	1	201	20,401	2,060,601	208,120,801
200	200	20,400	2,060,600	208,120,800	##############
Y	200	40,400	8,120,600	1,632,240,800	##############
	1	203	20,807	2,122,415	216,486,431
202	202	20,806	2,122,414	216,486,430	##############
	202	41,208	8,365,426	1,698,181,680	##############
B	1	205	21,217	2,185,453	225,101,761
204	204	21,216	2,185,452	225,101,760	##############
L	204	42,024	8,615,124	1,766,100,624	##############
	1	207	21,631	2,249,727	233,971,711
O 206	206	21,630	2,249,726	233,971,710	##############
	206	42,848	8,869,742	1,836,036,800	##############
C	1	213	22,897	2,450,085	262,159,201
212	212	22,896	2,450,084	262,159,200	##############
K	212	45,368	9,663,596	2,058,346,160	##############
	1	219	24,199	2,661,999	292,819,999
218	218	24,198	2,661,998	292,819,998	##############
	218	47,960	10,503,458	##############	##############
F	1	223	25,087	2,809,855	314,703,871
222	222	25,086	2,809,854	314,703,870	##############
A	222	49,728	11,089,566	##############	##############
	1	227	25,991	2,963,087	337,792,031
C 226	226	25,990	2,963,086	337,792,030	##############
	226	51,528	11,697,082	##############	##############
T	1	229	26,449	3,041,749	349,801,249
228	228	26,448	3,041,748	349,801,248	##############
O	228	52,440	12,008,988	##############	##############
	1	233	27,377	3,203,225	374,777,441
R 232	232	27,376	3,203,224	374,777,440	##############
	232	54,288	12,649,336	##############	##############
	1	237	28,321	3,370,317	401,067,841
236	236	28,320	3,370,316	401,067,840	##############
	236	56,168	13,312,052	##############	##############
	1	239	28,799	3,455,999	414,719,999
238	238	28,798	3,455,998	414,719,998	##############
	238	57,120	13,651,918	##############	##############
	1	255	32,767	4,194,303	536,870,911
254	254	32,766	4,194,302	536,870,910	##############
	254	65,024	16,581,374	##############	##############
	1	273	37,537	5,142,705	704,550,721
272	272	37,536	5,142,704	704,550,720	##############
	272	74,528	20,346,416	##############	##############
	1	277	38,641	5,371,237	746,602,081
276	276	38,640	5,371,236	746,602,080	##############
	276	76,728	21,253,932	##############	##############

TABLE A-2 MINIMUM/CRITICAL/MAXIMUM NUMBER OF KEYS FOR KEY BLOCKING FACTOR VS LEVEL

Min Crit Max	B-tree level 1	2	3	4	5
	1	281	39,761	5,606,441	790,508,321
K 280	280	39,760	5,606,440	790,508,320	##############
	280	78,960	22,188,040	##############	##############
E	1	291	42,631	6,224,271	908,743,711
290	290	42,630	6,224,270	908,743,710	##############
Y	290	84,680	24,642,170	##############	##############
	1	297	44,401	6,615,897	985,768,801
296	296	44,400	6,615,896	985,768,800	##############
B	296	88,208	26,198,072	##############	##############
	1	307	47,431	7,304,527	1,124,897,311
L 306	306	47,430	7,304,526	1,124,897,310	##############
	306	94,248	28,934,442	##############	##############
O	1	319	51,199	8,191,999	1,310,719,999
318	318	51,198	8,191,998	1,310,719,998	##############
C	318	101,760	32,461,758	##############	##############
	1	331	55,111	9,148,591	1,518,666,271
K 330	330	55,110	9,148,590	1,518,666,270	##############
	330	109,560	36,264,690	##############	##############
	1	341	58,481	10,000,421	1,710,072,161
F 340	340	58,480	10,000,420	1,710,072,160	##############
	340	116,280	39,651,820	##############	##############
A	1	357	64,081	11,470,677	2,053,251,361
356	356	64,080	11,470,676	2,053,251,360	##############
C	356	127,448	45,499,292	##############	##############
	1	383	73,727	14,155,775	
T 382	382	73,726	14,155,774	##############	
	382	146,688	56,181,886	##############	
O	1	409	84,049	17,230,249	
408	408	84,048	17,230,248	##############	
R	408	167,280	68,417,928	##############	

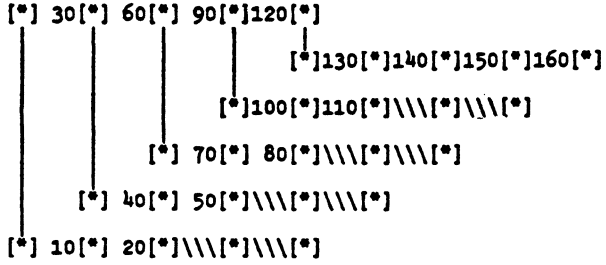
The minimum value is the absolute minimum number of keys that a B-tree can hold for a given level, if a key is deleted then the B-tree will contract one level. An example is shown in the following figure of a 2 level (L) B-tree with a key blocking factor (F) of 4 and with minimum number of keys.



The minimum number of keys (MinK) is given by the formula:

Mink = 2 * (F/2 + 1)^(L-1) - 1

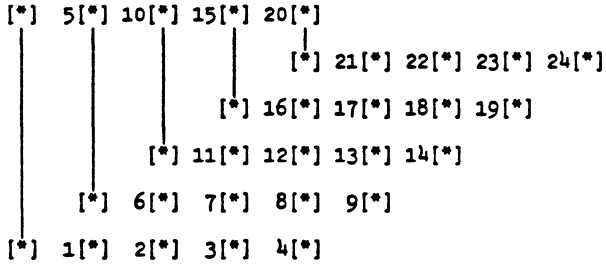
The 'critical' value is the minimum number of keys that the B-tree can hold on a given level before splitting and creating another level. This value is given by subtracting 1 from the minimum value of the next higher level.



The critical number of keys (CritK) is given by the formula:

CritK = 2 * (F/2 + 1)^(L) - 2

The maximum value is the maximum number of keys that the B-tree could hold if it would be possible to load it with no 'holes' whatsoever. An illustration of this concept is shown below on a 2 level B-tree with a key blocking factor of 4 and maximum number of keys.



The maximum number of keys (MaxK) is given by the formula:

MaxK = (F + 1)^(L) - 1

Appendix B - KSAM Worksheets

K S A M

DATAFILE SIZE WORKSHEET INSTRUCTIONS

A. Do part A.1 thru A.6 for the file. If the blocking factor is not specified then use 1 (one) which is the default for KSAM files.

B. This section includes the file system label and the KSAM created user label. The number of user labels in B.1 are in addition of the one created by KSAM. If no user labels are specified then enter 0.

C. To do part C requires knowing the the data file limit. If this value is not supplied then enter 1023 in C.1, since this is the default file limit for the data file.

KEYFILE SIZE WORKSHEET INSTRUCTIONS

Note that there are 2 formats for this worksheet. The first format is labeled 'PAGE 2 OF __' and the second format 'PAGE__ OF __' The first format includes space at the end of the page for introducing the totals of other pages in case the key file has more than 4 keys. Don't forget to label all the pages.

A. Do part A.1 thru A.3 for all the keys and A.4 thru A.12 only for those keys that have a Key Blocking Factor (KBF) specified.

B. Choose the largest value in A.12 and enter in all the columns in B.1. If none of the keys has a Key Blocking Factor (KBF) specified then enter 1024 in all the columns in B.1. Now perform all the steps in part B for all the keys to obtain the actual Key Blocking Factor.

C. To do part C requires knowing the number of key entries for the key file or the data file limit. If this value is not supplied then enter 1023 in all the columns in C.1, since this is the default file limit for the data file. The tricky part is C.7 where it represents the number of sectors per block (SB). This value can be obtained from the largest value found in A.10, but if none of the keys had a key blocking factor specified then this part was not filled in, and we had to use 1024 in B.1 for block size which is equal to 8 sectors. So in C.7 enter the largest value found in any A.10 column, or if there are no values in A.10 then enter 8 in C.7

D. Part D requires totaling all of the key chains and adding 3 for the normal key file overhead (1 for Control Block, 1 for the Key Descriptor Block and 1 for the File Label). At this point you are done!.

K S A M

PAGE 1 OF __

DATA FILE SIZE WORKSHEET

NAME _____ DATE __/__/__ TIME __:__

DATA FILE NAME _____._____._____ KEY FILE NAME _____

		FIXED	VARIABLE
A) SECTORS PER BLOCK			
1. RECORD SIZE (words)		_____	_____
1.1 ADD 1 (for variable only).....			+ 1
1.2 RESULT (for variable only).....			_____
2. MULTIPLY BY BLOCKING FACTOR		X_____	X_____
2.1 RESULT (for variable only).....			_____
2.2 ADD 1 (for variable only).....			+ 1
3. RESULT (words per block)	=	_____	_____
4. DIVIDE BY 128		/ 128	/ 128
5. R E S U L T		____.-	____.-
6. ROUND UP (sectors per block)	=	_____	_____
B) LABELS OVERHEAD			
1. NUMBER OF USER LABELS		_____	_____
2. ADD 2		+ 2	+ 2
3. R E S U L T	=	_____	_____
4. DIVIDE BY A.6		/_____	/_____
5. R E S U L T	=	_____	_____
6. ROUND UP (labels overhead)		_____	_____
C) FILE SIZE			
1. FILE LIMIT (FL)	=	(_____)	(_____)
2. DIVIDE BY BLOCKING FACTOR		/_____	
3. R E S U L T	=	____.-	
4. ROUND UP (# of blocks)	=	_____	
5. ADD B.6		+_____	_____
6. R E S U L T	=	_____	_____
7. MULTIPLY BY A.6		X_____	_____
8. ANSWER (SECTORS PER FILE)	=	_____	_____

K S A M

PAGE 2 OF __

KEY FILE SIZE WORKSHEET

NAME _____ DATE __/__/__ TIME __:__

DATA FILE NAME _____.____.____ KEY FILE NAME _____

		KEY 1	KEY 2	KEY 3	KEY 4
A) BLOCK SIZE					
1. KEY SIZE (wds)					
2. POINTER OVERHEAD		+ 4	+ 4	+ 4	+ 4
3. KEY ENTRY SIZE	=				
4. KEY BLOCK FACTOR (KBF)		X	X	X	X
5. T O T A L	=				
6. BLOCK OVERHEAD		+ 5	+ 5	+ 5	+ 5
7. KEY BLOCK SIZE	=				
8. DIVIDE BY 128		128	128	128	128
9. RESULT		___.__	___.__	___.__	___.__
10. ROUND UP (SB)					
11. MULTIPLY BY 128		X 128	X 128	X 128	X 128
12. ANSWER = WORDS/BLOCK	=				
B) KEY BLOCKING FACTOR					
1. LARGEST WORDS/BLOCK	=	(____)	(____)	(____)	(____)
2. BLOCK OVERHEAD		- 5	- 5	- 5	- 5
3. A N S W E R	=				
4. KEY ENTRY SIZE (A.3)		/___	/___	/___	/___
5. A N S W E R	=	___.__	___.__	___.__	___.__
6. ROUND DOWN	=				
7. MINUS 1		- 1	- 1	- 1	- 1
8. A N S W E R	=				
9. DIVIDE BY 2		/ 2	/ 2	/ 2	/ 2
10. A N S W E R	=	___.__	___.__	___.__	___.__
11. ROUND UP					
12. MULTIPLY BY 2		X 2	X 2	X 2	X 2
13. KEY BLOCK FACTOR (KBF)	=				
C) SECTORS/KEY-CHAIN					
1. DATA FILE LIMIT (FL)	=	(____)	(____)	(____)	(____)
2. DIVIDE BY KBF		/___	/___	/___	/___
3. A N S W E R	=	___.__	___.__	___.__	___.__
4. ROUND UP	=				
5. MULTIPLY BY 2		X 2	X 2	X 2	X 2
6. A N S W E R	=				
7. MULT BY LARGEST SB (A.10)		(X ___)	(X ___)	(X ___)	(X ___)
8. SECTORS/KEY_CHAIN	=				
D) KEY FILE SIZE		PAGE2	PAGE3	PAGE4	PAGE5
1. ADD ALL C.8	=				
2. ADD ALL D.1	=				
3. ADD 3		+ 3			
4. KEY FILE SIZE	=	_____ sectors			

K S A M

PAGE__ OF __

KEY FILE SIZE WORKSHEET

NAME _____ DATE __/__/__ TIME __:__

DATA FILE NAME _____._____._____ KEY FILE NAME _____

		KEY _	KEY _	KEY _	KEY _
A) BLOCK SIZE					
1. KEY SIZE (wds)					
2. POINTER OVERHEAD		+ 4	+ 4	+ 4	+ 4
3. KEY ENTRY SIZE	=				
4. KEY BLOCK FACTOR (KBF)		X	X	X	X
5. T O T A L	=				
6. BLOCK OVERHEAD		+ 5	+ 5	+ 5	+ 5
7. KEY BLOCK SIZE	=				
8. DIVIDE BY 128		128	128	128	128
9. RESULT					
10. ROUND UP (SB)					
11. MULTIPLY BY 128		X 128	X 128	X 128	X 128
12. ANSWER = WORDS/BLOCK	=				
B) KEY BLOCKING FACTOR					
1. LARGEST WORDS/BLOCK	=	( )	( )	( )	( )
2. BLOCK OVERHEAD		- 5	- 5	- 5	- 5
3. A N S W E R	=				
4. KEY ENTRY SIZE (A.3)		/	/	/	/
5. A N S W E R	=				
6. ROUND DOWN	=				
7. MINUS 1		- 1	- 1	- 1	- 1
8. A N S W E R	=				
9. DIVIDE BY 2		/ 2	/ 2	/ 2	/ 2
10. A N S W E R	=				
11. ROUND UP					
12. MULTIPLY BY 2		X 2	X 2	X 2	X 2
13. KEY BLOCK FACTOR (KBF)	=				
C) SECTORS/KEY-CHAIN					
1. DATA FILE LIMIT (FL)	=	( )	( )	( )	( )
2. DIVIDE BY KBF		/	/	/	/
3. A N S W E R	=				
4. ROUND UP	=				
5. MULTIPLY BY 2		X 2	X 2	X 2	X 2
6. A N S W E R	=				
7. MULT BY LARGEST SB (A.10)		(X)	(X)	(X)	(X)
8. SECTORS/KEY_CHAIN	=				
D) KEY FILE SIZE					
1. ADD ALL C.8	=				

ABSTRACT

Developing a DS To NS Network Migration Strategy

Diane Leeds, Hewlett Packard Company

NS 3000/V Network Services and links are an inices along with a discussion of enhancements to NS Services and LAN links and new link products that customers can use for wide area networking solutions.

In addition, a framework will be provided that can be used in planning this migration to take advantage of these new capabilities. This framework will present several alternative strategies, and the relative merits of each, that users can employ when planning their migration, taking into consideration the size and configuration of the network. Along with strategies, tools that are being developed to facilitate this migration will be discussed.

ABSTRACT

Artificial Intelligence? — Can We Use It To Solve Real Life Situations?

Mark Leiner, DOD Ft Meade

One of the agencies within the Department of Defense has undertaken the ambitious task of developing an Equipment Failure Reporting System for its many hi-tech maintenance and operational areas. This paper will try to answer the questions of what is artificial intelligence, and can we use this concept realistically is todays world? Specifically in this application, apply expert system technology to quickly diagnose equipment problems (much like an HP CE would do), thereby increasing productivity and reducing costs. Do our users want these benefits, if we can provide them, and if not, how do we convince them of its potential value?

Remote Data Collection With A Central Medical Data Base

Mary L. Lerchen and Ron Darling
New Mexico Tumor Registry
University of New Mexico
900 Camino de Salud NE
Albuquerque, NM 87131

Introduction

The purpose of this presentation is to: 1) describe our organization's successful computer conversion; 2) describe the software developed for cancer registries that we use on a Hewlett-Packard Series 42; and 3)discuss the development of the link between PCs and the mini computer and the flexibility it provides us.

First I will describe the New Mexico Tumor Registry (NMTR) role in data collection. The NMTR is a population based cancer registry which covers the entire population of New Mexico and All American Indians in Arizona. The NMTR participates in the Surveillance, Epidemiology and End Results (SEER) program of the National Cancer Institute, established to monitor cancer incidence and survival in selected areas of the United States. Abstractors visit over 90 facilities in New Mexico and Arizona to identify cases and to abstract demographic and medical information from hospital records. There have been approximately 70,000 persons diagnosed with cancer registered by the NMTR since the beginning of its operation in 1969. About 5,000 new diagnoses of cancer are registered each year among residents and added to the cancer data base. We follow about 30,000 persons annually to determine vital status for calculation of survival statistics. An example of summary statistics for cancer incidence and survival from New Mexico can be seen in Table 1.

Computer Conversion

When I joined the Registry in 1983 we had two major tasks ahead of us. One was to cut operating costs while improving quality and timeliness of data collection. The other was to decide whether to improve the existing computer system or to select another. After review of existing procedures and policies were were able to eliminate those that were duplicate or unnecessary efforts. After we redefined goals and streamlined procedures, the deficiencies of the computer system became more apparent. The limitations of the computer system that had been in place for three years are listed in Table 2. For example, we could not retrieve information from the data base; retrievals were run in batch mode by an off site computer. In addition, we were not able to match names by computer to determine if the cases were

already known to the Registry. Because of the high cost of hardware upgrade, the lack of software for data base management, and inaccurate information provided by the sales representatives, I decided to look at other systems. The selection of the Hewlett-Packard Series 42 minicomputer was determined by the availability of software designed for cancer registry applications. The staff at the Michigan Cancer Foundation (MCF) in Detroit, Michigan developed an on-line computerized system for inputting, editing and processing of data collected by registry staff. The MCF makes use of HP's data base and screen management software. Cobol programs serve as the interface between these packages. The data base software utilizes a network design and allows for access of data to be sequential, direct, or indexed and by batch or interactive mode. Because the computer software for the Cancer Information Management System (CIMS) was developed with funds from the National Cancer Institute, the system is available free of charge under the Freedom of Information Act. Should you be interested in the system, the contact person is: Michael Baracy, Manager Computer Systems, Michigan Cancer Foundation, 110 E. Warren, Detroit, Michigan 48201.

Conversion Timetable and Cost

The conversion from the existing computer system to the HP Series 42 involved several steps, often occurring concurrently, that can be seen in Table 3. The easiest step was to install the HP computer and software in November. In November and December we sent programmers for training to HP and to Detroit. Beginning in October and continuing through January we compared data elements collected by the Detroit Registry with those we collected. We also reviewed forms and procedures used by Detroit. Whenever possible we modified New Mexico to match Detroit so that changes to software programs could be kept to a minimum. The third step, conversion of existing data base, took place over a weekend in February without any problems. At the same time program modifications were completed that allowed data entry, edit, link and inquiry on the HP. The existing computer was used only to generate follow-up inquiries to physicians and patients; maintenance on the machine was discontinued. Once the "heart" of the CIMS was running, i.e., essential data collection and processing functions were in place, we could proceed at a more relaxed pace with the other processes such as casefinding and follow-up. In addition to a data and program conversion that progressed on time and without major problems, we can report that costs association with the conversion were reasonable (see Table 4). The only cost associated with acquiring the Detroit software was for computer tapes and for travel to Detroit to evaluate their program. Programmer training for the HP com-

puter was about $5000. Consultant costs included salary for the analyst with HP experience who worked less than 50% time for us throughout the conversion. Non equipment costs for conversion were only about 16,000. We are very satisfied with the CIMS, with the HP computer, and with the cost of the conversion.

The Cancer Information Management System

In describing the CIMS used at the NMTR, I will first list the hardware and HP software, then show flow diagrams for the CIMS, and finally discuss the timetable and costs for conversion. The hardware and software now in use at the Registry is shown in Table 3. We separated wordprocessing from the minicomputer before the conversion. We initially linked only one personal computer with the data base on the mini; the flexibility the PC provide for data collection, downloading and data analysis will be discussed later.

The flow diagrams for the data bases are seen in Figures 1-7. The main data base within the CIMS system is the Active data base. There are six keys within the Active data base to allow for quick access in locating records and are used extensively in the record linkage process and requests for data retrieval and reporting. These six keys are represented by the triangles in figure 1.

The six keys are date of birth, social security number, primary cancer site, year of diagnosis, NYSIIS name encription, and case number. The case number is a sequential number with a check digit and is the unique identifier assigned to each patient in the system and serves as the logical link between all the detail data sets.

The main structure of the Active data base includes four detail data sets containing Patient, Tumor, Admission, and Treatment specific information. These data sets are represented by the trapezoids in figure 1.

There is one record maintained in the Patient data set for each patient containing demographic information. The Tumor data set contains specific information for each primary malignant tumor that a patient has along with consolidated treatment information. The Admission data set contains information about the first admission to a specific hospital for each primary malignant tumor. Finally, the Treatment data set contains information describing all first course treatment that a patient received for each primary malignant tumor. First course is defined as all treatment received within the first four months after treatment was started.

There are five other data sets that are part of the Active

data base. These data sets are used for reporting, editing, verification of incoming data. The AKA data set is the "also known as" name data set which contains all assumed first and last names. The Address data set contains the address of a patient when they were diagnosed, but only if the address is within Bernalillo county. The Table Master data set contains ICD-O topography and morphology codes, site group codes, birth place code, abstractors identification numbers, and county codes. The Doctor Master data set contains specific information about every currently practicing doctor within New Mexico. Lastly, the Hospital Master data set contains information about all of the hospitals, clinics, and therapy facilities where information is abstracted.

A second data base is the Suspense data base. This data base contains newly keyed records not edited or ready for addition to the Active data base. The Suspense data base has two primary data sets, the Initial Admission data set and the Follow-Up Admission data set, they are represented by the trapezoids in figure 2.

Five keys are used to access these two data sets. The keys are document identification number, social security number, NYSIIS name encription, date of birth, and case number and are represented by triangles in figure 2.

The Initial Admission data set contains information abstracted from the hospital records. The Follow-Up Admission data sets contains follow-up information received concerning patient status (alive or dead), additional treatment given, and other information concerning subsequent admissions for the patient.

CIMS has several major processes. The most important of these processes are Abstract Data Entry, Linkage, Doctor File and Table Maintenance, Death Tape Match, SEER Tape, Follow-up, and Case Finding.

Initial and follow-up documents are keyed on-line into the Suspense data base and a comprehensive set of edits are performed in the Abstract Data Entry process. These documents are then added to the Active data base in the Linkage process with matching performed between new documents and old cases. In the Doctor File and Table Maintenance process information on doctors, hospitals, morphology codes, ICD-O topography codes, county codes, and more are maintained. Death tapes from the State of New Mexico are used to update the Active data base through the Death Tape Match process and are a major source of information. All cases in the Active data base that meet the SEER Program requirements are

written to a tape and sent to the National Cancer Institute in the SEER Tape process. Lastly, in the Case Finding process printouts are generated of cases that need to researched and abstracted to then be keyed in the Abstract Data Entry process.

All of the batch entry and much of the file maintenance is done on-line with various levels of edits performed to insure data integrity. However, batch processing is used to perform major updates to the various data bases. Table 6 lists interactive programs available on CIMS for data inquiry and maintenance. Table 7 show the batch processes used for updating and reporting and the type of report associated with each program.

Personal Computers Linked to CIMS

In December of 1986 we committed ourselves to the idea of data collection and analysis with personal computers. Various software packages are available that allow the abstractor to key abstracted information at the PC. In addition, this software allows data retrieval and analysis on the PC in the hospital setting. Advantages over abstracting onto forms include autocoding, simple edits and screen prompts that replace instructions in manuals. Although PC software was available, no one had yet linked the PC to a data base to that data could be transferred in both directions (both to the central registry and to the hospital) as is shown in Figure 8. We wanted to transmit cases to CIMS to take advantage of sophisticated edits and the matching processes and to transmit edited cases back to the hospital tumor registry so that data could be analyzed on site. The software we are using to collect and analyze data on a PC is based on the Hospital PC System written in dBase III by Charlie Smart, MD, while at the Utah Cancer Registry. We have made extensive changes to the Utah software to make it compatible with the CIMS software on the HP 3000 and we have named our version PC DaSH. PC DaSH stands for Patient Cancer Data System for Hospitals. These changes relatively easy to make thanks to dBase III. The software can be obtained for a nominal charge and the contact person is: Rosemary Dibble, Manager of Operations, Utah Cancer Registry, Research Park, 420 Chipeta Way, Suite 190, Salt Lake City, Utah 84108.

In setting up a hospital on PC DaSH the first step is to give the software and a hospital's old cases to the hospital, refer to figure 9. Then on a weekly or bi-weekly basis the person at the hospital downloads the new cases to a floppy diskette that they have abstracted during the past week or two. Someone at the central registry then loads these cases into the CIMS Suspense data base where they go

through the regular Edit and Link process. These cases and any old cases that have been updated are then sent back to the hospital to re load into their PC. Refer to figure 10 to see the flow of this process.

Summary

The intent of this presentation was to tell a success story about our computer conversion. We were successful for a number of reasons. First, we selected software carefully and the software did what it was supposed to do and was well written. And the NMTR had great support from the analyst who had written the programs at our sister registry in Detroit. Second, we had good support from HP in the selection, installation, and maintenance of the computer and software. And third, we had a dedicated staff that worked hard to bring about a success.

AVERAGE ANNUAL CRUDE,AGE-SPECIFIC, AND CUMULATIVE (AGE 0-74)
INCIDENCE RATES PER 100,000 POPULATION MALIGNANT CASES BY PRIMARY SITE,

NEW MEXICO RESIDENTS, ANGLO, FEMALES, 1983-1985

PRIMARY SITE	CRUDE RATE	<5	5-9	10-14	15-19	20-24	25-29	30-34	35-39	40-44	45-49	50-54	55-59	60-64	65-69	70-74	75+	CUM. 0-74
ALL SITES	354.5	11.9	6.4	12.9	16.6	30.5	61.4	130.3	172.6	300.2	346.8	490.0	619.4	890.9	1105.0	1365.6	1812.0	27.8
BUCCAL CAVITY & PHARYNX	8.5	...	1.3	...	...	2.0	...	1.1	1.4	9.4	8.1	8.0	25.9	16.0	43.9	21.1	38.1	0.7
LIP	1.4	...	...	...	...	...	...	...	...	1.6	...	...	3.1	...	10.0	2.6	12.7	0.1
TONGUE	1.3	...	...	...	...	2.0	...	...	...	3.1	1.6	1.6	3.1	1.8	2.0	7.9	3.6	0.1
MAJOR SALIVARY GLANDS	1.0	...	...	...	...	...	...	1.1	1.4	1.6	3.2	3.2	1.5	...	2.0	...	5.4	0.1
FLOOR OF MOUTH	1.3	...	...	...	...	...	...	...	...	...	...	1.6	4.6	1.8	16.0	2.6	1.8	0.1
GUM AND OTHER MOUTH	1.7	...	1.3	...	...	...	...	...	...	1.6	1.6	1.6	4.6	5.3	2.0	5.3	10.9	0.1
NASOPHARYNX	0.1	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	1.8	...
TONSIL	0.6	...	...	...	...	...	...	...	...	1.6	...	...	4.6	1.8	4.0	...	...	0.1
OROPHARYNX	0.3	...	...	...	...	...	...	...	...	...	1.6	...	1.5	...	2.0	2.6	...	0.0
HYPOPHARYNX	0.7	...	...	...	...	...	...	...	...	...	...	...	3.1	3.6	6.0	...	1.8	0.1
PHARYNX & OTHER NEC	0.1	...	...	...	...	...	...	...	...	...	...	...	...	1.8	...	...	...	0.0
DIGESTIVE SYSTEM	66.9	...	...	...	...	1.0	1.0	4.5	13.6	12.5	35.5	49.5	111.4	162.1	195.5	292.6	574.4	4.4
ESOPHAGUS	1.6	...	...	...	...	...	...	...	...	...	...	...	6.1	7.1	8.0	5.3	7.2	0.1
STOMACH	5.5	...	...	...	...	...	...	...	4.1	1.6	...	11.2	6.1	7.1	14.0	21.1	52.5	0.3
SMALL INTESTINES	1.0	...	...	...	...	...	...	...	1.4	...	1.6	...	1.5	1.8	...	7.9	7.2	0.1
COLON EXCLUDING RECTUM	33.1	...	...	...	...	1.0	...	3.4	2.7	6.3	16.1	28.7	47.3	78.4	75.8	166.1	300.8	2.1
RECTOSIGMOID JUNCTION	3.1	...	...	...	...	...	...	...	...	1.6	6.5	1.6	7.6	10.7	6.0	5.3	23.6	0.2
RECTUM	9.1	...	...	...	...	...	...	...	2.7	1.6	3.2	1.6	21.4	19.6	33.9	31.6	79.7	0.6
ANUS, CANAL & ANORECT	1.0	...	...	...	...	...	...	...	...	...	1.6	...	4.6	1.8	2.0	10.5	3.6	0.1
LIVER	1.0	...	...	...	...	...	1.0	...	...	...	...	...	3.1	1.8	4.0	5.3	7.2	0.1
GALLBLADDER	1.6	...	...	...	...	...	...	...	...	...	...	...	1.5	5.3	6.0	5.3	16.3	0.1
OTHER BILIARY	1.4	...	...	...	...	...	...	1.1	...	...	...	...	...	5.3	8.0	5.3	10.9	0.1
PANCREAS	7.7	...	...	...	...	...	...	...	1.4	1.6	6.5	6.4	9.2	23.2	35.9	26.4	56.2	0.6
RETROPERITONEUM	0.3	...	...	...	...	...	...	...	1.4	...	...	...	...	...	2.0	2.6	...	0.0
PERITONEUM	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
OTHER DIGESTIVE ORGANS	0.6	...	...	...	...	...	...	...	...	...	...	...	3.1	...	...	...	9.1	0.0
RESPIRATORY SYSTEM	39.1	...	...	...	...	1.0	...	1.1	6.8	14.1	17.7	55.9	79.3	149.7	201.5	184.5	143.1	3.6
NASAL CAV, SINUS, EAR	0.4	...	...	...	...	...	...	...	1.4	...	...	...	1.5	...	...	...	5.4	0.0
LARYNX	1.5	...	...	...	...	...	...	...	...	...	1.6	3.2	10.7	3.6	6.0	2.6	1.8	0.1
LUNG AND BRONCHUS	36.8	...	...	...	...	1.0	...	1.1	5.4	14.1	16.1	52.7	67.1	146.1	193.5	179.3	132.3	3.4
TRACHEA	0.3	...	...	...	...	...	...	...	...	...	...	...	...	...	2.0	2.6	3.6	0.0
BONES AND JOINTS	0.9	1.3	1.3	1.2	1.0	1.0	...	1.1	...	1.6	1.6	...	...	...	...	2.6	1.8	0.1
SOFT TISSUES(INC HEART)	2.7	1.3	2.6	...	...	1.0	2.0	1.1	2.7	4.7	...	1.6	3.1	5.3	8.0	2.6	14.5	0.2
SKIN(EXC BASAL & SQUAM)	17.1	...	...	...	...	1.0	13.3	34.0	23.1	26.6	25.8	19.2	29.0	26.7	23.9	23.7	63.4	1.2
MELANOMAS OF SKIN	16.6	...	...	...	...	1.0	13.3	32.8	21.7	26.6	25.8	19.2	29.0	26.7	21.9	23.7	58.0	1.2
OTHER SKIN CANCERS	0.5	...	...	...	...	...	...	1.1	1.4	...	...	...	...	...	2.0	...	5.4	0.0
BREAST	108.2	...	...	...	1.0	3.0	9.2	26.1	63.9	139.2	166.1	201.1	198.3	265.5	333.1	419.2	425.8	9.1
FEMALE GENITAL SYSTEM	51.8	...	...	...	...	11.2	17.4	26.1	28.5	42.2	51.6	83.0	80.9	160.4	155.6	218.8	193.9	4.4
CERVIX UTERI	10.6	...	...	...	...	5.1	10.2	19.3	20.4	20.3	11.3	16.0	12.2	21.4	12.0	23.7	18.1	0.9
CORPUS UTERI	22.6	...	...	...	...	...	1.0	2.3	1.4	9.4	19.4	28.7	41.2	99.8	85.8	116.0	88.8	2.0
UTERUS, NOS	0.1											1.6						0.0

Table 2

Deficiencies of the Old Computer and Software

- Did not have data base management or data base manager for the creation of reports or online retrieval.
- Flow of the system was extremely complicated, and was understood only by the programmer who wrote the system.
- No matching capabilities.
- Alot of computer down time, with slow and unpredictable service.
- Expensive to maintain.
- Software worked in batch mode only.
- Data entry screens where very hard to design and code.
- Upper limits of the CPU had been reached without a major upgrade.

Table 3

Timetable for Conversion to the
HP 3000 Computer System and CIMS

Decision to convert to CIMS	March 1985
HP System Installed	November 1986
Programmer Training (HP and CIMS)	November & December
System Review and Design	October - January
Data element review	
IMAGE schema changes and related data base changes	
Accounting structure and security	
Conversion of existing data	February
Program Modifications	
Data entry	February
Edit module	February
Link and update module	February
Suspense data base maintenance	February
Active data base maintenance	February
Data base inquiry process	February
(the above processes are the "heart" of CIMS)	
Table maintenance process	March
Conversion of reporting packages	March
Case finding	March
Patient Follow-up programs	April
SEER tape processing	July
Death linkage and update	August
PC DaSH Link to CIMS	
Develope network	October - January 1987
Test link in 2 hospitals	January - March
Key data using PC DaSH	June
Data retrieval & analysis in hospital	July

Table 4

Costs Associated with Conversion to CIMS

Hewlett Packard 3000/42 (includes all hardware & software)	$125,000
CIMS Programs	cost of tapes
HP Training	5,000
Travel between Detroit and Albuquerque (Program evaluation and training)	2,300
Consultants	9,000
Personnel - Two programmers (on staff) and 25-50% time analyst (consultant) November through February. One programmer and 50% analyst (consultant) March and April	
Total	$141,300
Cost minus equipment	$16,300

Table 5

Equipment List

Hewlett Packard 3000 Series 42
- 2 Megabytes of Main Memory
- 1 404 Megabyte Disc Drive
- 1 571 Megabyte Disc Drive (Recent Addition)
- 1 1600 BPI Tape Drive
- 1 300 Line Per Minute Printer
- 14 Block Mode Display Terminals

Personal Computers
- 6 HP Vectras with 20 Mb Hard Drives
  (also tied to the HP 3000/42 above)
- 4 IBM XTs with 10 Mb Hard Drives (word processing)
- 8 HP LaserJet, ThinkJet, QuietJet, and Epson Printers
- One H.P. Eight Pen Plotter

Software

Hewlett Packard 3000 Series 42
- MPE Operating System
- IMAGE Data Base Management System
- V/3000 Screen Communications
- COBOL II
- Fortran
- Editor
- Query/3000
- Inform/3000, Report/3000, and Dictionary/3000
- Toolset
- Cancer Information Management System (CIMS)

Personal Computers
- Wordstar 3000
- Lotus 1-2-3
- dBase III Plus
- Graphics Gallery
- AdvanceLink 2392
- Easyflow

Table 6

Interactive Programs
For
Data Inquiry and Maintenance

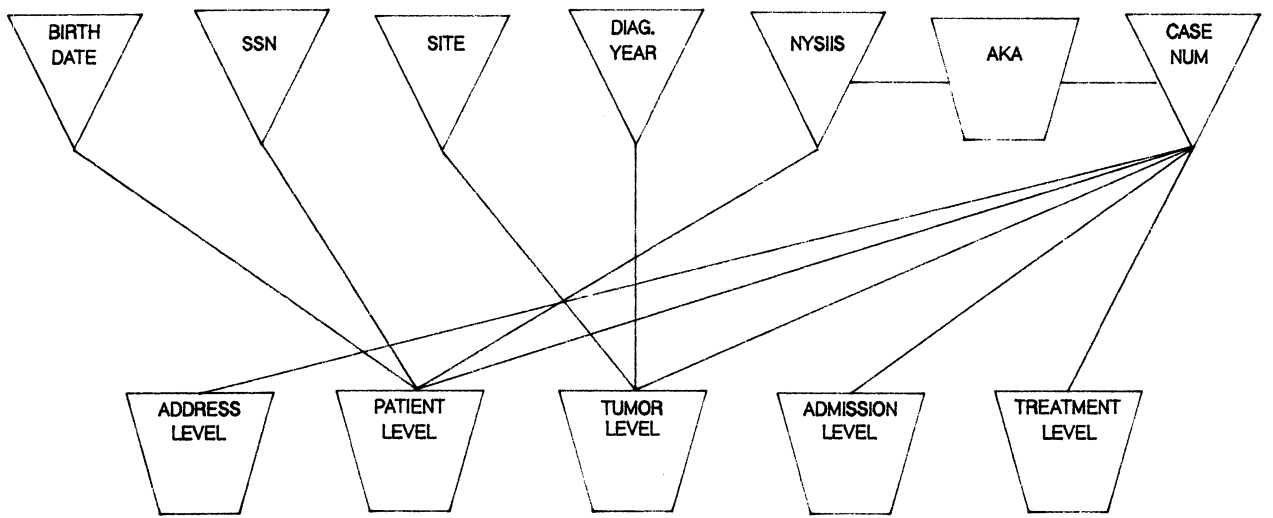
Program	Type of Edit
Abstract Data Entry	Intra-Record
Case Finding Entry	Intra-Record
Suspense Data Base Maintenance	Intra-Record
Inquiry and Matching	
Active Data Base Maintenance	Comprehensive and Inter-Record
Table Maintenance	Intra-Record

Table 7

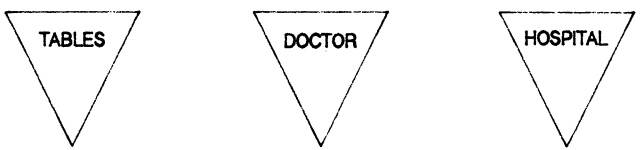
Batch Processing
For
Updating and Reporting

Process	Reports
Edits	Cases with errors
Linkage	Inconsistencies, possible and perfect matches
Death Match	Audit report, possible matches
SEER Tape	Cases with errors
Case Finding	Case finding lists, alpha lists
Follow-Up	Physician and patient letters, Hospital lists
Management & Administrative	Cases by facility, by year, etc.

Figure 1
CIMS1 – Active Data Base
Key Access Paths



Tables



Remote Data Collection With a Central Medical Data Base

Figure 2
CIMS3 - Suspense Data Base
Key Access Paths

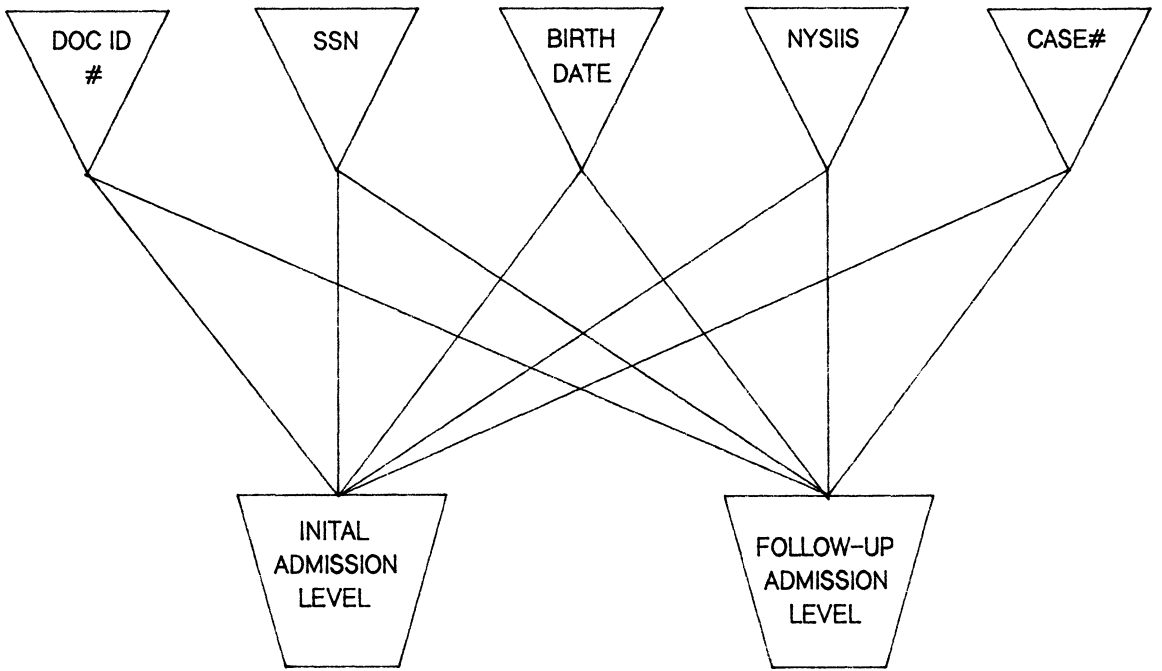
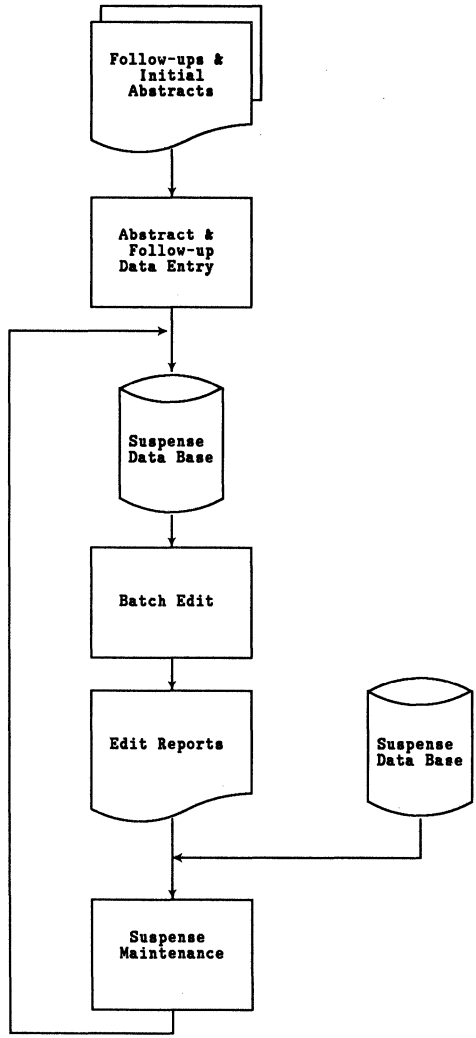


Figure 3

Cancer Information Management System (CIMS)

Abstract Data Entry

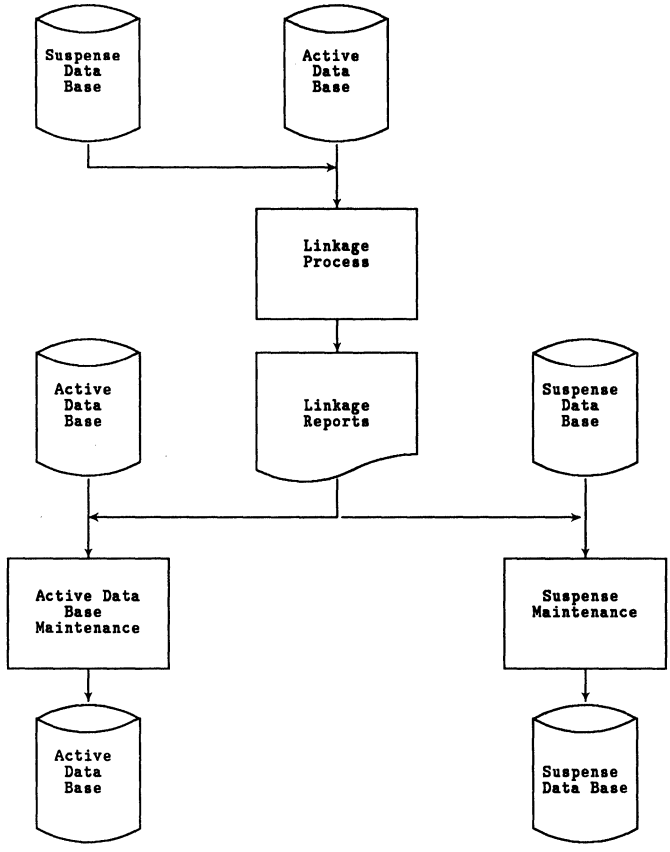


Remote Data Collection With a Central Medical Data Base

Figure 4

Cancer Information Management System (CIMS)

Linkage Process

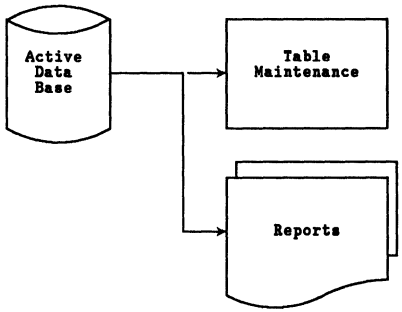


Remote Data Collection With a Central Medical Data base

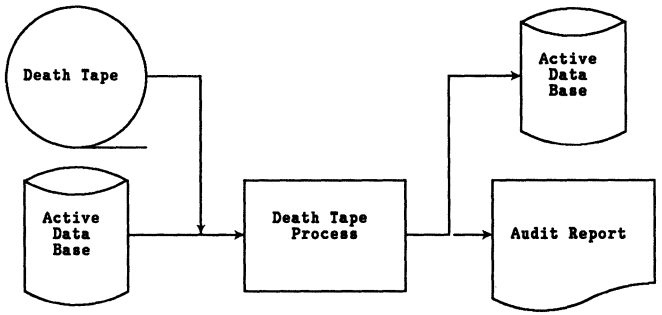
Figure 5

Cancer Information Management System (CIMS)

Doctor File and Table Maintenance



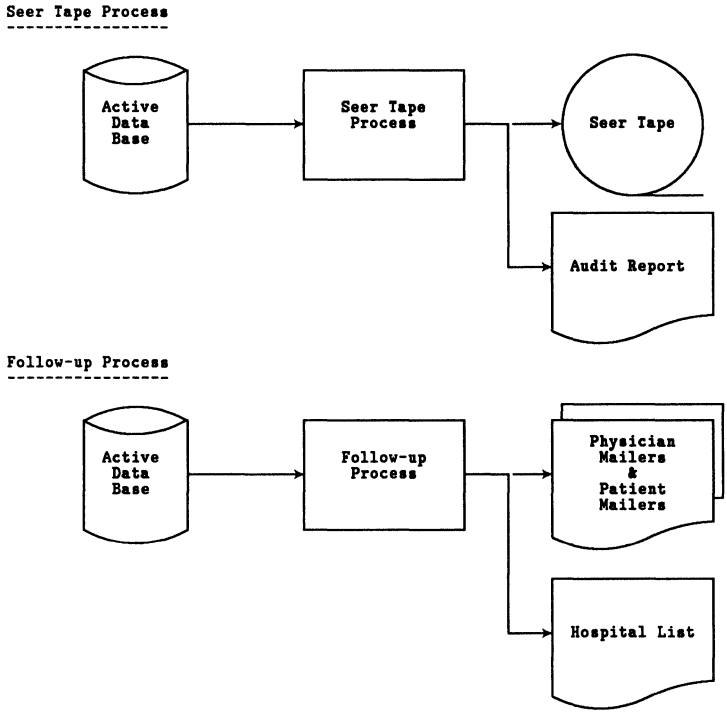
Death Tape Match



Remote Data Collection With a Central Medical Data Base

Figure 6

Cancer Information Management System (CIMS)

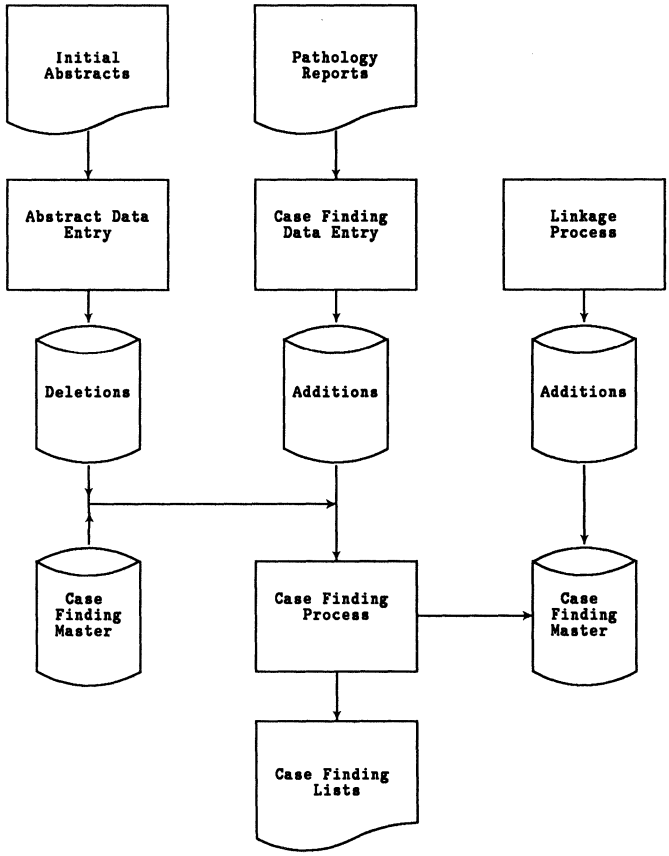


Remote Data Collection With a Central Medical Data Base

Figure 7

Cancer Information Management System (CIMS)

Case Finding Process



Remote Data Collection With a Central Medical Data Base

Figure 8

Relationship Between Hospitals and Central Registry

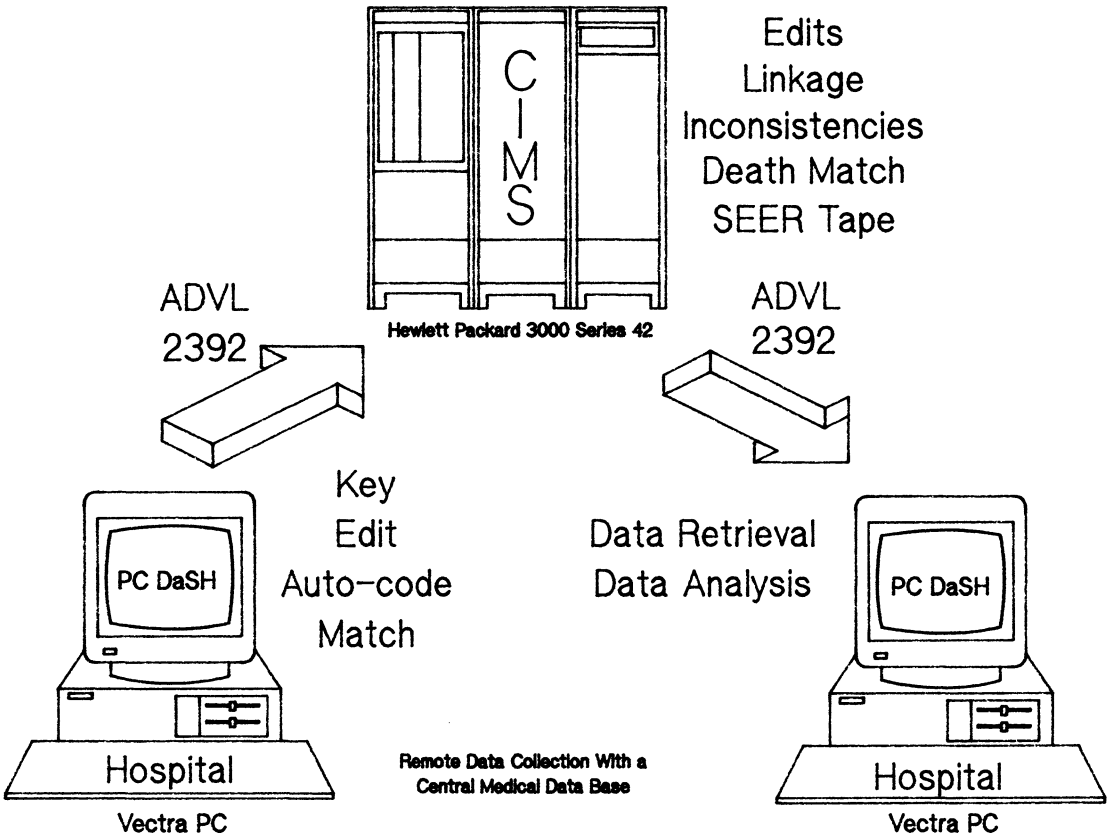
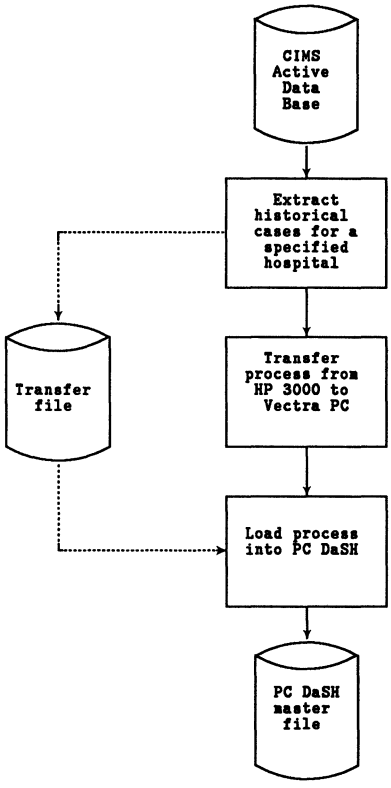


Figure 9

CIMS to PC DaSH

Down Loading Historical Data

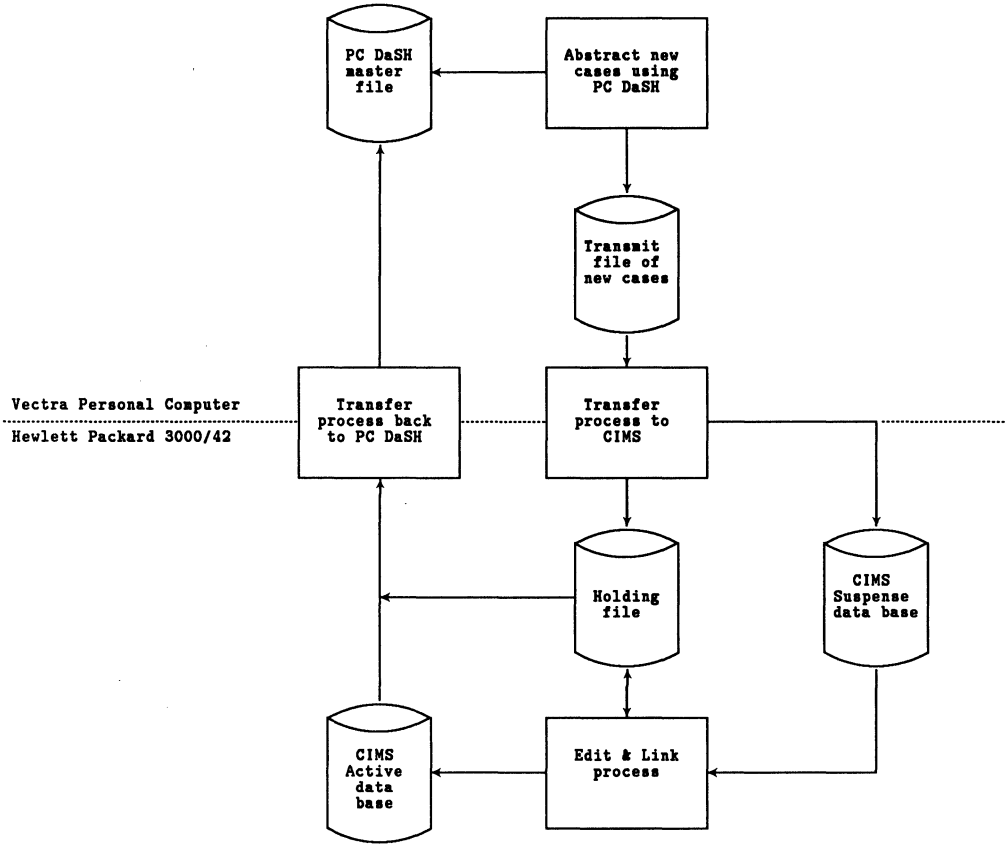


Remote Data Collection With a Central Medical Data Base

Figure 10

CIMS to PC DaSH

Cyclical Flow of Data on a Bi-Weekly Basis



Remote Data Collection With a Central Medical Data Base

A Skeleton in the Closet

by

Kenneth W. Lessey
Marjorie K. Lessey

DataCon of St. Helens, Inc.
50 West Street
St. Helens, ORegon 97051

INTRODUCTION

Users assume that computer programmers have and use software development technology that allows us to produce perfect, or nearly perfect, programs. They assume that when we author a program we know what the program will do under various conditions, and that we have verified that the program actually performs the intended functions. However, most programs will execute differently at different times, based on the data that they encounter. Amazingly, software developers seldom make a comprehensive list of these different functions, so we cannot possibly have proven that all of them work correctly. This is the skeleton in our closet. No proof can ever exist using current development technologies.

It has always been possible to prove that a program does not work by executing it, but recognizing that a program can be proven to work correctly is a new concept to programmers - even though our users have been assuming that we were doing it all along. If the methodology you currently use does not allow you to prove that your software works perfectly, then the methodology is inadequate, because the users expect it to work perfectly.

In our office, we have been using one methodology that recognizes the problem that software must be proven. We have taken several steps to address the need to "prove" software before it reaches the user, and we find that our users are very happy with the end results.

PATHS

It is possible to visualize a program as a sequence of operations occurring one after the other. The word "path" is defined as the "sequence of events" in a program, and a path may be very complex with multiple branches (offering multiple routes from the beginning to the end). The street map for any town offers a good example of a complex path. If we wish to get from one side of the town to the other, we have many options from which to choose. The particular path we take will be affected by the time of day, the day of the week, local traffic and local business activity. These varibles, taken together, will determine the success of our endeavor. We may arrive safely, but too late for our appointment. The overall result in the above example is failure, even though we crossed town successfully.

Currently used programming techniques progress from designing a program to testing in order to find bugs and correct them. We then deliver the "finished product" to the user. Continued use of the program normally defines more bugs which are then corrected, so that over time, the program will mature into a reliable condition. Unfortunately, many programs have become obsolete by the time they have matured and are considered "reliable".

To actually prove that a program works reliably, it is necessary to define every path that can be taken, and then demonstrate that the total result of taking that path produces the desired result. Traditional programming technologies do not allow us to identify all the possible paths, so our testing procedures, at best, allow us to prove that bugs exist. The identification and exercise of all paths is a giant part (now missing) of developing reliable software.

It could be argued that the use of non-procedural languages for software development provides a set of predefined and tested paths. In truth, the companies that have developed and currently market, procedural languages do not know, and have therefore not tested, every path. An additional problem occurs when a path we need does not exist or an existing path does not produce the desired result. All of these problems make the non-procedural languages even riskier to use. A widely known horror story about the problems we can encounter with non-procedural languages is the story about a contractor who submitted a bid on a large project based on dollar amounts totalled by his spreadsheet program. Only after having been awarded the contract was the (giant) error discovered. The error, it turns out, was actually caused by a bug in the spreadsheet program, but the contractor ended up taking the loss because he was forced to honor his bid.

A methodology for developing reliable software does exist, but there is a price to pay. In order to implement a development technology that addresses the entire process, from program design, through path identification and testing, requires us to relinquish some of our current practices. Given the fact that old software systems die daily, either because they are not reliable, or because they are not maintainable, I submit that we have no choice but to switch to newer methodologies. Given the fact that many programmers

"would rather fight than switch", this topic will not delight everyone.

TRANSITION NETWORKS

History shows us that many great inventions were not based on totally new ideas. Rather, existing facts came to be understood in a new way. In our search for ways to make our software more reliable, we happened on to a new perspective in our use of transition networks which has provided a successful methodology for us to use in software development.

It has often been said "A picture's worth a thousand words". We have, for many years, attempted to include a picture of a program's flow of logic in the program file. These help us to quickly understand what is happening (and where) when we come back to a program to enhance or correct it. These diagrams are even more essential in our office because we frequently have someone other than the original author assigned to add an enhancement. We have used transition networks to create these pictures.

In a traditional state diagram, the sequence of events is graphically represented by circles or nodes connected by lines. These diagrams show functions as being performed on the lines, and decisions are made at the nodes (where to branch to next).

SEE Figure 1

As our staff worked with these diagrams, we discovered that they often confused, rather than clarified, our understanding of the sequence of events. In retrospect, we have decided that these diagrams are similar to attempting to draw road maps using lines to represent cities and nodes to represent all the branching roads between them. At best, they are confusing to use to graphically represent the flow of logic through a program.

We have developed a modified type of transition network which works well for us. We represent states, or functions that we perform, as squares. Lines are drawn to point the way to the next function to be performed.

SEE Figure 2

To the casual observer, this technique may appear to be similar to a flow chart. But do not condemn the use of transition networks because of preconceived ideas you may have about flow charts. The only similarity between the two techniques is that both use symbols connected by lines. The use of transition networks provides a way to define every possible path through a program, so if you are sincerely interested in "proving" your software, you will be forced to find a way to define your paths. Transition networks have provided an easy way for our staff to do that. (There have been several good computer science textbooks published on the techniques used to produce transition networks.)

TESTING

The third step in the process of designing reliable, provable software (after code design and path identification) is to design test data which will direct the program down every path, and execute the program using that test data. I can well imagine that those of you who have been open-minded enough to consider everything we have presented so far just stomped on the brakes, and cried, "Enough!". It seems to most programmers that a test designed to exercise a complex program on every conceivable path just adds insult to injury. It is bad enough to have to test some of these "monsters" at all, and the kind of testing we are proposing looks like a giant task, surrounded by frustration, and based in futility. After all, some paths may never be used, even by live data! Be assured that the depth of testing we propose is the only way to deliver reliable mature software (that is maintainable).

Our staff discovered that we frequently spent more time testing programs than we spent in the design and coding phase. We have also discovered that very few of us actually enjoy the testing cycle (test-fix-test again) which is sometimes repeated countless times. Our answer has been to automate the testing as much as possible.

In order to automate the testing cycle, we use a color coded copy of paths on the transition network. We then design a set of data which will force the program down each path. We call this data a scenario, or script, and each script is made up of many individual steps. We store the test data in a separate data base. This allows us to execute a script to prove the paths after a

program has been modified or enhanced in any way. We then examine the results for correctness. In truth, we have not yet been able to ship truly "bug free" software to our users, but we have been able to eliminate most of the bugs by using an automated testing system. We have discovered that there are a few laws which govern this type of testing.

1. Test data must be stored in a data base (in order to be re-used and modified).
2. The actual execution of a scenario needs to be automated - this is to save the sanity of programmers.
3. The larger and the more complex the system, the more important this testing is.

CONCLUSION

In the past, commonly used software development methodologies could best be described as "hit and miss", because they did not contain ALL of the key elements needed to design reliable software. Today's users demand reliability, so we must use methodologies which are capable of producing the programs (in all their complexities), and have, as an inherent attribute, a "way to prove" that you did it. The second thing we must do is to implement the proof.

At least one methodology does exist which contains the necessary elements to produce reliable, maintainable software which can be proven. Appendix A contains the detailed description of how we use transition networks, and Appendix B contains information on our automated test system, ENSIGN. (In our office, we also believe that good software must be capable of changing as company needs change.)

Transition networks, in combination with entity relationship analysis for data base design (see papers by Ray Thomas(1) and Alfredo Rego(2)) and automated testing allow us to produce software that meets user's expectations. If you are an MIS manager and your shop chooses to use a methodology which does not include a way to "prove" your software, the skeleton in the closet may be be yours. (This paper will self-destruct if it finds its way into the hands of any user.)

EXAMPLE

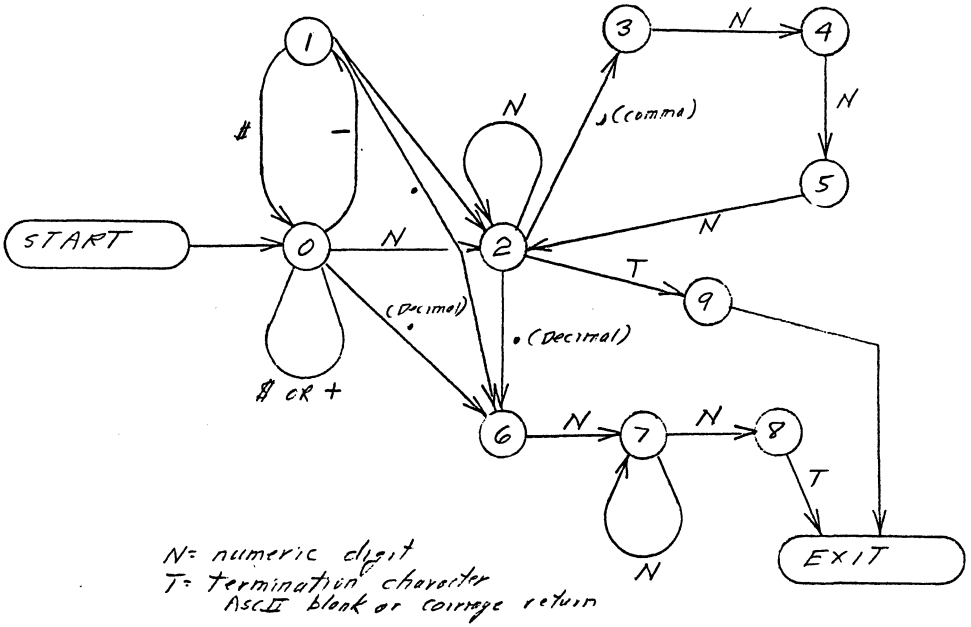


Figure 1

EXAMPLE

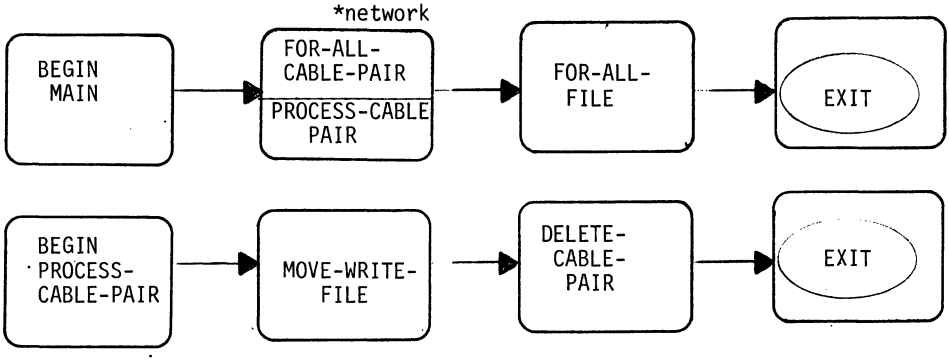


Figure 2

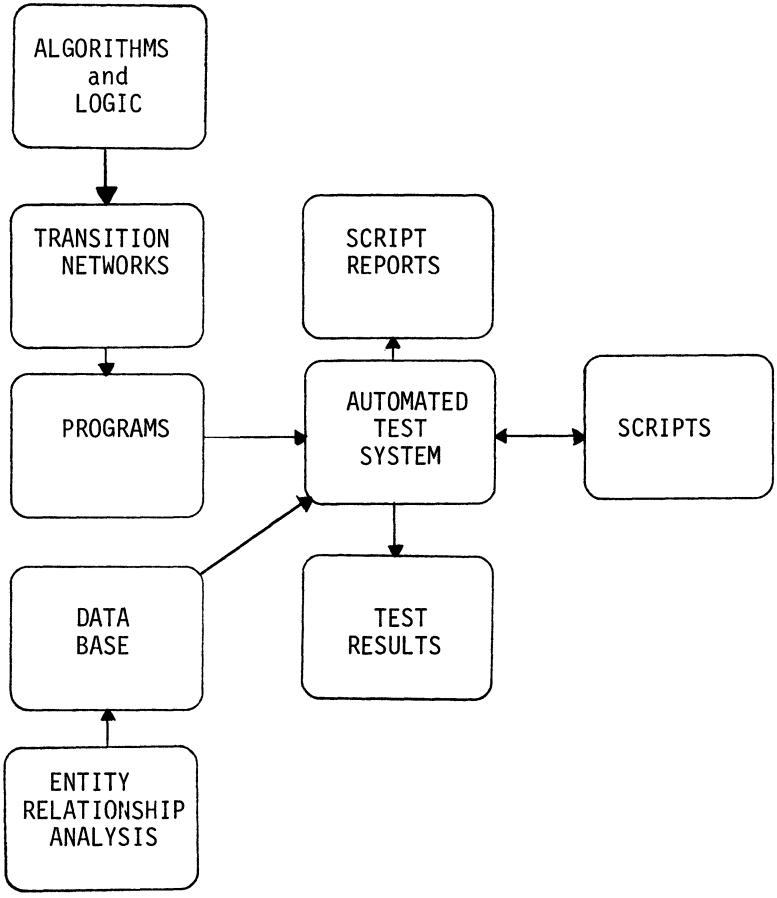
FOOTNOTES

1. "Entity Relationship Analysis, A Methodical Method for Implementing Relational Data Base on the HP 3000 with IMAGE," Ray Thomas of Texas Municipal Power Agency, Published in the Montreal Proceedings, HPIUG, April 1983.

2. "Are Procrustean Databases Necessary?", F. Alfredo Rego, Published by Supergroup Association, Volume 5, Issue 1, January/February 1985.

APPENDIX A

DEVELOPMENT METHODOLOGY OVERVIEW



One definition of a program is "a sequence of operations occurring one after the other". In order to produce a picture of the sequence in which operations occur, we have developed a way to graphically represent a program. We use these pictures for testing as well as for writing the code.

While transition networks have been around for years, they have not been widely used. We think that there are several reasons for this lack of use. One reason is that a traditional use of transition networks presents operations as lines, and decisions as nodes (where to branch to next). Our experience has taught us to use the opposite combination. In other words, we represent an event (or operation) by a box, and we show where to go next by directed lines between the boxes. The traditional networks are difficult to follow, therefore, few people are willing to make the effort. Another reason that traditional transition networks may be of questionable value for our purposes (path definition) is that decisions, as represented by nodes, can direct the program to any of several paths, depending on the data. Our experience has taught us to refine decisions, and break complex questions down until each question can be answered "true" or "false".

In using transition networks to diagram the flow of logic through our programs, we have come up with some fundamental rules we follow.

1. We all use the same approach, so that any one of us can understand (and therefore work on) a program written by any other staff member. Standardization is important in order to optimize time spent on enhancement and testing.

2. We developed pre-printed forms to draw logic networks, and we use pencils (and large erasers) to make it easy to modify the diagrams.

3. A box can represent either a single state, or it can represent complex logic (which must be diagramed elsewhere state by state). The use of this technique allows us to diagram any degree of complexity because a network can execute another network which can execute another network, etc.

allowing multiple depth levels for your logic.

4. We require that questions be answered by "true" or "false" only. Any question too complex for a true/false answer must be divided into a series of questions, each of which is answerable by true or false only.

5. When we start a logic diagram, we map logic from the highest level (overview) possible. We then diagram the logic at lower and lower levels until all logic has been mapped. See Figure 1-A for an example overview diagram and one of the lower level diagrams. Note that we mark those boxes which represent another network which is diagramed in detail elsewhere.

6. As we begin to write the actual code for each state, we sometimes encounter the need for logic which exceeds the level of complexity we allow in a single state. When that happens, we simply design a logic network to replace that single state, mark the square on the diagram as a network, and add a complete diagram of the new logic to the network pages for that program.

When we have completed the design phase and the actual code writing functions, we return to the diagram to identify possible paths through the program.

We identify the paths through each network individually, and then design our test data to force the program to execute every path. See Figure 2-A for the possible paths through the example network (shown in Figure 1-A). We actually mark each path with colored pens because the different colors allow us to easily follow a trace to see that each path has been executed correctly. (For ease in printing, we have identified the paths using different symbols.) We normally mark the longest possible path as the first path to take with test data, and then take the next shorter path, etc.

It may occur to you that the use of transition networks and the testing phase we are proposing will add to the effort required to produce a "finished" program. We agree, and these are the alternatives as we see them:

1. you can by-pass this diagram/path identification step and (in effect) let your user test the

software. You then fix it, and present it to the user to test again, and so on ad infinitum.

2. you can take the time and make the effort to "prove" a program and then deliver it to the user. This is the end of this process in most cases, unless the user wants to modify specs.

We have found that by choosing alternative 2, we by-pass a large portion of user frustration. In the long run, we save a great deal of programmer frustration as well, and "they all lived happily ever after".

This example comes from the logic to gather the data from the data base to print a report.

FIGURE 1-A

This example comes from the logic to gather the data from the data base to print a report.

FIGURE 2-A

APPENDIX B

ENSIGN, an automated testing system, allows us to exercise programs thoroughly before they are delivered to users. In order to automate testing, it is necessary to have a system by which you can build and maintain scripts, and it is vital that you build the capability to execute scripts into all programs. We will use our own testing system as an example.

We store the data used to exercise a program in a separate data base. See Figure 1-B for a picture of the ENSIGN data base. The data sets have been named to make it obvious what kinds of records each contains. (The prefix "EN" merely conforms to our internal naming conventions, and stands for ENSIGN.)

```
DATA SET:  EN-SCRIPT                    D   DATABASE ENSIGN
   KEYS    EN-SCRIPT-A          EN-SCRIPT-U
DATA ITEMS:
   EN-SCRIPT-U                  X(30)
   EN-SCRIPT-A                  S9(9) COMP
```

We use this data set to contain the names of our scripts. The actual name is stored in the element EN-SCRIPT-U, and a numeric value is assigned to each name which is stored in the element EN-SCRIPT-A. We have defined a script as a group of steps to be executed consecutively.

```
DATA SET:  EN-SCRIPT-COM                D  DATABASE ENSIGN
   KEYS    EN-SCRIPT-A
DATA ITEMS:
   EN-SCRIPT-A                  S9(9) COMP
   EN-COMMENT                   X(78)
```

We use this data set to store comments which give information required to execute a script. These comments give specific directions for testing a program correctly. The element EN-SCRIPT-A is the numeric identification which ties the text stored in the element EN-COMMENT to the correct script.

```
DATA SET:  EN-SCRIPT-STEP              D  DATABASE ENSIGN
   KEYS    EN-SCRIPT-A          EN-STEP-A
DATA ITEMS:
   EN-SCRIPT-A                  S9(9) COMP
   EN-STEP-A                    S9(9) COMP
   EN-SEQ-NBR                   9(6)
```

This data set is used to establish the link between a script and the individual steps which it contains. The element EN-SEQ-Nbr is used to store the sequence of this step in the script.

```
DATA SET:  EN-STEP                     D  DATABASE ENSIGN
   KEYS    EN-STEP-U            EN-STEP-A
DATA ITEMS:
   EN-STEP-U                    X(16)
   EN-STEP-A                    S9(9) COMP
```

We have defined a step as an individual action, and each script is made up on many steps. The element EN-STEP-U is used to store the identification for the step, and EN-STEP-A is used to store the numeric identification which has been assigned to that step.

```
DATA SET:  EN-STEP-COMMENT             D  DATABASE ENSIGN
   KEYS    EN-STEP-A
DATA ITEMS:
   EN-STEP-A                    S9(9) COMP
   EN-COMMENT                   X(7889)
```

This data set is used to store comments which give information about a step. The element EN-STEP-A contains the numeric identification of the step to which it applies, and the element EN-COMMENT contains the actual comment. We often use this element to store a message we expect to see on the terminal when this step has been executed in a script.

```
DATA SET:  EN-MAN-ACTION               D  DATABASE ENSIGN
   KEYS    EN-STEP-A
DATA ITEMS:
   EN-STEP-A                    S9(9) COMP
   EN-SEQ-NBR                   9(6)
   EN-FUNCTION-KEY              S9(4) COMP
   EN-COMMENT                   X(78)
```

This data set is also used to store "comment" information about a step, but the information stored here gives the tester instructions to perform some manual action, such as pressing a function key, at this time, before proceeding to the next step.

The element EN-STEP-A contains the numeric identification of the step affected by this information. The element EN-SEQ-NBR us used to store the sequence in the script where this action is to take place. The element EN-FUNCTION-KEY is used to store the number of the function key which is to be pushed, and the element EN-COMMENT is used to store any explanation necessary to successfully complete this manual action.

```
DATA SET:  EN-SCREEN-BUFFER              D  DATABASE ENSIGN
   KEYS    EN-FORM-A            EN-STEP-A
DATA ITEMS:
   EN-STEP-A                    S9(9) COMP
   EN-FORM-A                    S9(0) COMP
   EN-DATA-BUFFER               X(255) OCCURS 6 TIMES
```

This data set is used to store a copy of the fields off the screen (form from formspec). EN-STEP-A contains the numeric identification of the step which uses this screen (form) and the element EN-FORM-A contains the numeric identification of form itself. The element EN-DATA-BUFFER contains a mirror image of the form (the occurs statement as part of the picture statement allows us to use a field large enough to contain the image of the screen).

```
DATA SET:  EN-FORM                       D  DATABASE ENSIGN
   KEYS    EN-FORM-U            EN-FORM-A
DATA ITEMS
   EN-FORM-U                    X(16)
   EN-FORM-A                    S9(9) COMP
   EN-FORM-FILE                 X(36)
```

This data set is used to store the name of the form (in EN-FORM-U), the numeric identification of the form (in EN-FORM-A), and the name of the formfile (in EN-FORM-FILE).

In order to maintain the data in the scripts, we have written a screen program to Add, Show, Change and Delete data for each of the data sets. We have also written a

report program which prints out a script. The report program incorporates the comments associated with each step, as well as the script comments. We keep a hard copy of each script in the file for a program. The comments give us extra help in executing a script. See Figure 2-B for a sample portion of a printed script. (We have also written a "fix" program which is an automated way to fix the screen buffer in the event that a client decides to change a screen. We have used it plenty.)

With the above data base, screen programs and reports in place, we have a system which allows us to build and maintain scripts. In order to execute the scripts, we needed to insert logic to activate (or not activate) the data stored in the ENSIGN data base. A system like ENSIGN can be added to the logic of virtually any program. Figure 3-B shows the network diagram for the logic we insert at the beginning of each program. Figure 4-B shows examples of the code in each state included in the network diagram.

When the time comes to test a program using a script, we first set a JCW called ENSIGNON to the value 1. The code we add to each program first checks to see if ENSIGN is active, and if it is, the program reads script data from the ENSIGN data base and displays the data for the first step on the terminal. The program then waits for a human to push the enter key. Once the enter key has been pushed, the program executes normally (in relation to the data that was on the screen), and then reads the data for the next step from the ENSIGN data base, displays it on the terminal, and waits for someone to push the enter key, etc. Because we use a JCW to activate the testing cycle and a separate data base for test data must be on the system, it is unlikely that a user would inadvertently activate the test mode. ENSIGN is basically transparent to the user.

There are undoubtedly many different ways to implement a system to automate the testing cycle. The important point is that the testing cycle must be automated before systematic and comprehensive software testing becomes an accomplished fact.

ENSIGN DATA BASE

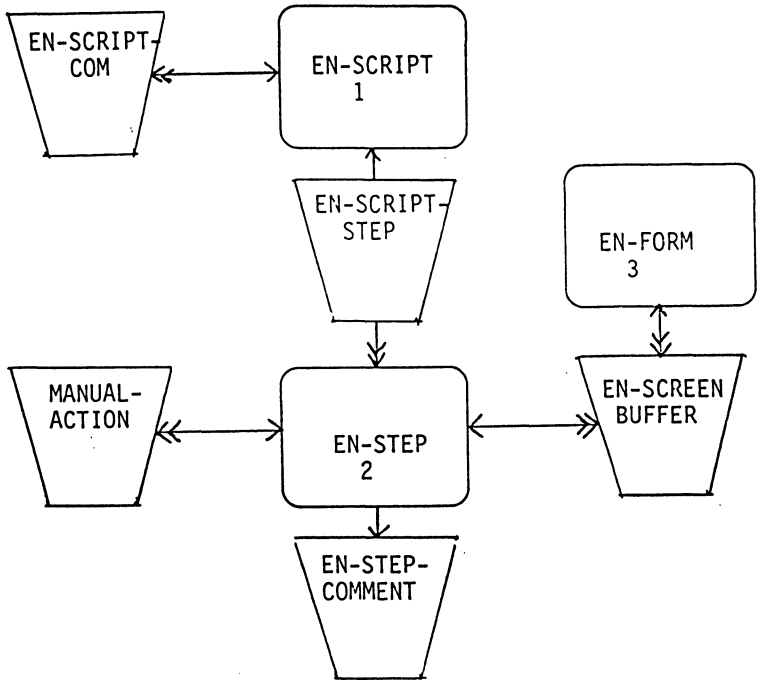


Figure 1-B

Script SERORDCSD

The SERORDCSD script cannot be run alone. You must run the SERORD02 script first. The instructions for the two scripts are in comments for SERORD02. This script cannot edit a change on Service Orders already posted.

```
************************* Step    10  Form SERVICE-ORD-F    ******************

S                        JMG00012C    30
Message: entry is shown

************************* Step    20  Form                  ******************

Sequence    1  Push      Next Step

************************* Step    30  Form SERVICE-ORD-F    ******************

C3979999IN022886      JG JMG00012A35    0228863977775              2Y966822449
Message: can not change phone number

************************* Step    40  Form SERVICE-ORD-F    ******************

C3977775IN022886         JMG00012C    40022886                   2Y966822449
Message: must have entry in this field

************************* Step    50  Form SERVICE-ORD-F    ******************

C3977775IN022886      JG JMG00012C    50022886                   2Y966822449
Message: bad house number

************************* Step    60  Form SERVICE-ORD-F    ******************

C3977775IN022886      JG JMG00012C    60022886                   2Y966822449
Message: bad street name

************************* Step    70  Form SERVICE-ORD-F    ******************

C3977775IN022886      JG JMG00012C    70022886                   2Y966822449
Message: bad even digit

************************* Step    80  Form SERVICE-ORD-F    ******************

C3977775IN022886      JG JMG00012C    80022886                   2Y966822449
Message: bad odd  digit

************************* Step    90  Form SERVICE-ORD-F    ******************

C3977775IN022886      JG JMG00012C    90022886                   2Y966822449
Message: data was changed

************************* Step   100  Form SERVICE-ORD-F    ******************
```

FIGURE 2-B

FIGURE 3-B

Skeleton

```
BEGINSTATE ENSIGN-ACTIVE
   IF ENSIGN-ON-FLAG = "Y" AND
      V-LASTKEY = 0
      SET-TRUE
   ENDIF
ENDSTATE

BEGINSTATE GET-NEXT-EN-STEP
   WITH EN-SCRIPT-STEP
   BEGIN
   GET NEXT EN-SCRIPT-STEP
   ON ERROR
      SET-TRUE
   END
ENDSTATE

BEGINSTATE GET-EN-STEP
   BEGIN
   GET EN-STEP WITH EN-STEP-A = EN-SCRIPT-STEP.EN-STEP-A
   END
ENDSTATE

BEGINSTATE GET-EN-SCREEN-BUF
   WITH EN-SCREEN-BUFFER
   BEGIN
   GET EN-SCREEN-BUFFER WITH EN-STEP-A =
                        EN-SCRIPT-STEP.EN-STEP-A
   ON ERROR
      SET-TRUE
   END
ENDSTATE

BEGINSTATE MOVE-DATA-SCREEN-BUF
   WITH EN-SCREEN-BUFFER
   BEGIN
   FOR EN-I = 1 TO 6
      MOVE EN-DATA-BUFFER(EN-I) TO EN-BUF-1(EN-I)
   ENDFOR
   END
ENDSTATE

BEGINSTATE DISPLAY-TRAN-STEP
   WITH EN-SCRIPT-STEP
   BEGIN
   MOVE SPACES             TO TRAN-Z
   MOVE EN-SEQ-NBR         TO EN-STEP-Z
   DISPLAY TRAN-Z EN-STEP-Z
   END
ENDSTATE

BEGINSTATE TALK-TO-SCREEN
   MOVE TRAN               TO SAVE-TRAN
   READ
ENDSTATE
```

FIGURE 4-B

Application Data Exchange - Beyond File Transfer.

Richard Linnett.
Cognos Inc.
Ottawa. Canada

Introduction.

File transfer between the HP3000 and a microcomputer is an increasingly popular tool allowing users to integrate host data into spreadsheets and documents.

There are problems with simple file transfer however, from the question of how to define the required extract, or even of how to let a user know what data is available.

Products are available, not only for HP3000, but for the entire gamut of mini and mainframe computers, that provide controlled access and transformation to industry standard formats.

A new need is emerging however, for access to data independent of location. To provide an adequate solution requires programatic access from an application on the micro to data at the host, and that poses challenges beyond those addressed by existing file transfer solutions.

This paper examines the problems we have encountered implementing generalised access to HP3000 data from a micro, and outlines the solutions we have adopted in building closely integrated solutions.

Where are we now?

Since the introduction of microcomputers into the office environment, there have been requests for a means of transferring data from the central HP3000 to the microcomputer. Although only two or three years ago these requests were infrequent, they are now extremely common.

As an example, we find that many companies have developed a series of extracts based on the following model:

- Use QUIZ to extract, format and summarise required data from the corporate database, saving the report output to a disk file.
- Use a terminal emulation program to transfer the data down to the PC.
- Use a BASIC program to convert what is now a text file into the target format - for example a PRN file, a comma delimited file, or a DIF file.
- Load the data into the PC application.

This approach works quite well in production situations where the same format file is transferred often, but because of the effort required to build the transformation programs, is unsuitable for adhoc requests.

What is needed.

The limitations of file transfer and transformation have forced us to look closely at the problem. The advent of the Cognos product line on PCs, has increased the need for an effective solution.

We see that in the next few years, that the following will occur:

- The PC will cease to be a standalone machine, but will become a fully integrated workstation in an overall DP environment. Data exchange between the PC and the HP3000 will become the norm.
- Rather than upgrading a central HP3000 system, users will opt for an HP3000/Vectra solution that utilises the power of the both the HP3000 and the PC. Suppliers will be expected to provide connectivity between the two.
- Simple file transfer will not be enough. That effectively batch process will need to be replaced by interactive data exchange between programs on the PC and the HP3000.
- The connection will become a pipeline not only for data, but also for distribution of new or updated programs and applications.
- With the increasing transfer of information between machines, the bottleneck of asynchronous communications will be overcome by a major increase in the use of LANs.
- 4GL and other software products will evolve to provide a seamless distribution of applications between the PC and the HP3000.
- The corporate mainframe will not be allowed to remain isolated, but will be incorporated into the network, giving the much discussed 'three tier' solution. Access to mainframe data will in many cases be through the minicomputer, direct access from micro to mainframe being disallowed.

A Practical Experiment

The prospect of designing and building a complete solution is daunting, many problems remain to be solved and much work is required before even prototype systems can be in place.

We feel that the best approach is to adopt an incremental implementation plan with clear intermediate levels that lead eventually to full "three tier" integration.

In order to demonstrate the possibilities of a distributed solution, The first implementation that came to mind was to provide something close to file I/O redirection - a call level library on the PC to transfer I/O requests to the HP3000 where a slave process would call the actual intrinsics. This approach has its attractions, after all this is surely the concept of virtual disk drives and the like.

We built a prototype distributed system to this I/O redirection model, and found much to no ones surprise, a number of problems:

Data communications was a major problem:

- Although the future is definitely with LANs, asynchronous is still the present, and an asynchronous link between a PC and an HP3000 is a very slow process.
- No message transfer software was available off the shelf, so we had to develop a series of routines to pass data between the PC and the HP3000. We had to get involved in all the detail of data communications, the HP3000 proving a most unforgiving host!
- As everyone discovers, text data transfers reasonably, but binary data transfer causes problems. A control Y or an ENQ/ACK code for example, would be interpreted by the HP3000 as control codes, not only being lost to the program, but directly effecting the link as well. The problem is easily solved - replace the problem codes with escape sequences before sending the data and translate back at the receiving end, the cost being in throughput.

Beyond data communications, many more problems surfaced:

- We needed to develop routines to transform integer values between the two systems by switching the byte order. Floating point conversion was far more complex, and what to do with data types that were unique to one machine was not immediately obvious.
- The problems of application version control are significant. Just what does happen if the HP3000 database is changed but not all copies of the PC program that update it?
- Data integrity is also a major headache. The update of an Image database at the same time as a KSAM file, and maintaining integrity is difficult enough, but the problem becomes far more significant when the files are on two isolated machines.
- Very few standards appeared to exist for interprocess communications. Although the prototype worked in our environment, the question always remained, what would happen if our link had to coexist with some other product?
- For our experiment I/O redirection at the intrinsic level was adequate, but would it be acceptable to the user community as a whole? Would a programmer at the micro accept coding Image intrinsic calls?
- The model also proved totally inadequate when it came to satisfying the security requirements of a general purpose link. Security at a file level is totally inadequate when controlling data that is to be downloaded to a PC. A much finer level of control is required.
- A further problem is in finding out what data is available on the HP3000. At the PC, a user will often just scan through files until the right file is found - a similar approach on the HP3000 would be totally unacceptable. Some directory of available data is required.
- Performance was a major issue. Time to transfer data across a 9600bps link was significant, and simple file I/O redirection requires a lot of data to be moved. In addition the duration of Locks was much greater when the locks were invoked from a remote program - and although we did achieve some reduction in processing at the HP3000, the maximum number of concurrent users dropped considerably.

In summary, it was reasonably easy to implement the file I/O model and although it was slow, it worked. The solution was not without problems however, and we feel that it would be unwise to use this model in any major production effort.

What Comes Next?

We have been able to use the prototype to verify our thinking and establish a framework for what will be required of application environments in a distributed world.

we have to address the problems of performance. Unless performance can approach that obtained when running a standalone PC application, the user will not easily accept a distributed application. A cursory glance at the prototype system shows that the major factor in throughput is data transfer across the communications link. *We have to reduce communications traffic to a minimum.*

Simplifying HP3000 file structures can be achieved by allowing the user to *view all data on the remote HP3000 as if it were part of a relational dbms.*

The use of a relational model has a number of advantages:

- A full relational interface allows complex data manipulations to be defined across the link. By having the HP3000 perform all selections, joins, sorts accumulations and the like, only a small result set is transferred across the link. This in itself providing considerable reductions in the volume of data transferred.
- The relational paradigm removes from the PC, and more importantly the PC user, the need to understand a number of complex foreign file systems. If mainframe or 'foreign minicomputer' file systems are added to the picture, this independence becomes even more important.
- The relational model actions data at an item level, not at a record level. It is this level that is required for adequate security controls on data access.
- By providing the user with a location independent view of the data, we must expect that data requests will span several systems. Intelligent analysis of a global request to provide the most efficient sub requests becomes essential.

A strong dictionary on the HP3000 does become essential. It must allow MIS to control access to the data as well as providing the mapping between the relational model and the real world of Image and KSAM.

Even a simple foray into data communications, like was necessary for our prototype system, gives an appreciation of the need for adopting a data communications standard. *Unfortunately there are a number of standards to choose from.* As an example we have:

- A NETBIOS/MSNET level interface, very appropriate for connecting a PC into a LAN. However, look beyond the simple PC interconnect problem - would these be a suitable standard between HP3000s?
- TCP/IP already with a considerable following. The upcoming link from TCP/IP to NETBIOS, this would provide a solid interface across many different machine environments.
- IBM have their own set of rules, (or in this case sets of rules). The most current being APPC/SRPI.
- The OSI 7 layer standard protocol, increasingly popular in Europe.

Any of the above standards could become the norm, but what is more likely to happen is that these standards will continue to coexist - each within its own particular sphere of influence.

Rather than waiting for a dominant standard to appear, we have found it necessary to adopt the following approach:

- No application will interact directly with the communications environment, but will use the equivalent of an IBM APPC call level interface.
- A number of fairly intelligent interface routines, providing at the low level, calls to a particular communications standard are needed. This will allow the required interface to be selected at run time.

Transformation of data types is simple if we have simple record structures, well behaved programs and only Vectras and HP150s to contend with.

However:

- Other machines, particularly the Apple Macintosh and 680x0 based Unix systems are becoming more popular.
- Data files do not have simple record structures, complex redefined records have always been around.
- Many programs assume alignment and bit or byte ordering in a data record - structure that does not hold true across machine architectures.

It becomes necessary to adopt a complete data transformation strategy if future problems are to be avoided. One defacto standard exists, the External Data Representation (XDR) developed and released to the public domain by Sun Microsystems, provides a means of encoding data independently across a link between any number of heterogeneous systems. This standard does solve the problem of data transformation between any number of systems but it is not a total panacea - the program that makes assumptions about structure may still give invalid results.

Application currency is another topic of extreme importance. As we move away from central operations, the basically manual methods now used to control implementation of new versions of an application are no longer adequate.

The problem extends to all aspects of a distributed application. We must anticipate not only duplication of programs, but also data files and dictionary information as well.

Admittedly the use of virtual disks to hold central copies of an application makes it easier to control, but in reality *we no longer have control over the version of an application the end user will try and use.*

The application itself will have to provide the required controls over product versions used. This could be left entirely to the programmer, requiring the exchange of some control messages that identify the application level. More likely however, is an environment where the underlying software provides a complete set of controls that not only verify currency but will also replace out of date versions where required.

The process is extremely simple in concept, yet there are problems if a global resource such as Dictionary 3000 is used, that would preclude the use of simple 'date last updated' algorithms to recognise an incorrect version. This type of problem needs to be solved if complete version control is to be made available. However, at a minimum, the software MUST recognise correct version(s) of a program, preventing use of any incorrect version.

Data integrity is a major concern. Problems exist when updating across KSAM files and an Image database on the same HP3000. *The problems are magnified in a distributed environment:*

- If the application data exists across a number of PCs as well as on the HP3000, how can the data be guaranteed to be correct?
- If data is replicated on a number of systems across the network in an attempt to improve performance, how is the data guaranteed to be current at all times?
- What happens if the data on the HP3000 is updated, but an error occurs on the PC before data there is fully updated, will data be out of sync between the two machines?

Some database systems do use a technique known as two phase commit to control updates between databases (which may be distributed across machines), but they work strictly within the bounds of the database files - KSAM and MPE file updates are not included.

A practical approach must be adopted if integrity is to be maintained, as follows:

- Data updates have to be performed within strong transaction boundaries, with all changes within a transaction applied correctly, or none applied. *Applications cannot be allowed to make unconstrained updates across file systems.*
- Partial transaction completion must be avoided at all costs.
- Where possible, The techniques of two stage commit must be applied across the network to ensure compatibility between data on the different machines.

This approach is not foolproof, and it may cause some performance degradation. However that must be weighed against the problems caused by the corporate data becoming corrupt. What is likely is that many applications will be built that place restrictions on where updates can be performed, disallowing cross machine updates within a single transaction.

Summary

We, like most software suppliers, have been working on integration of the PC into the corporate HP3000 world.

We believe that although the hardware vendors may well provide the essential connectivity, it will still require considerable effort to integrate systems into a seamless user environment.

This seamless integration will require many of the features that are now found in a 4GL:

- A Strong data dictionary will be needed to identify available data, to define the location of data, to transform the data between machine specific syntax, and to provide necessary controls over access.
- Extensive recovery mechanisms will be required to ensure integrity of data not only between file types, but across heterogeneous computer systems.
- Equally complete mechanisms will be needed to support the integrity of the application itself. New versions of application programs, table files etc must be propagated automatically across the network.
- The application language for building distributed applications will, of necessity be easy to use - many users will be non technical micro users performing ad hoc requests. However, the language will also need to be extremely powerful to support programmers wishing to build true distributed applications.

PC integration is arriving - it may be painful over the next few years with incompatibility between products causing problems. However, once the marketplace settles down the potential is great:

- The usefulness of the PC will increase substantially as data from corporate systems becomes available to the PC application.
- By giving the user access to corporate data, through simple PC type products, demands on MIS resources for simple adhoc extracts could be reduced.
- The HP3000 will be offloaded considerably, much of the processing required for presentation of user data being done where it belongs - on the PC.

In the long run, both user and MIS manager will benefit.

SIMULATING RELATIONAL DATABASES USING IMAGE AND OTHER CURRENTLY AVAILABLE SOFTWARE

Kerry Lloyd
System Manager, HP3000
System Automation Corporation
8555 16th Street
Silver Spring, MD 20910

INTRODUCTION

Much attention has been drawn lately to the subject of relational databases. Essentially, a relational database allows the rapid extraction of desired data from a number of related and unrelated datasets into tables from which desired reports can then be drawn. Speed and ease of programming the queries from both the original dataset grouping and the extracted tables are deemed to be hallmarks of a relational database.

The IMAGE database available on HP3000 computers is a networking database, and as such has the capability to simulate relational databases given proper design and the use of some currently available software items such as 4GLs and database manipulators. While IMAGE may not be the fastest DBMS in existence, its capabilities are extensive and, properly utilized, can provide an acceptable substitute for a newer software technology.

This paper endeavors to describe some working principles which will provide such a capability using IMAGE. Principles are discussed in general, but a example database and programs for handling a football pool have been provided to be made available on the CSL tape. Since the 4GL in use at the author's company is POWERHOUSE, concrete examples within the paper, and including the pool database, are provided in that language; two database manipulators or handlers (DBH), ADAGER and DBGENERAL, are also in use, and references to DBHs relate to either of these products.

DEFINITION OF TERMS

Several relatively new terms and others commonly heard will be used frequently in the course of this paper, so it is only fair to provide the reader with definitions.

Volatile Data — data which is changed frequently; records are added or deleted on a regular basis, or the data in a given record or set of records is updated on a repeating but relatively short schedule. Detail files such as invoice headers and line–items or payroll information collected on a weekly basis fall into the volatile data category.

Static Data — data which changes infrequently if ever; files are used for lookup or contain master data records which do not require update. Files containing information such as company addresses, meanings of codes used for lookup, etc., are the prime members of the static data category.

Extract Table — a set of records containing the desired data to be used to produce one or several reports. Such data has been culled or combed from the database records and put into a separate file. The speed of reporting is considerably improved when reports are produced from extract tables.

DB Handler — (DBH) a software package which allows the appropriate users to "fiddle" with the database, in terms of adjusting capacities, adding or deleting paths, items, and datasets, etc. Examples of DBHs include ADAGER and DBGENERAL; several others are available but the author has only had a small amount of exposure to these other packages, and does not feel qualified to comment on them. All handlers should share certain capabilities in common, including path manipulation and item and dataset creation and modification.

4GL — (4th Generation Language) a procedural language which allows users to develop applications for database usage in appreciably shorter time than the

3rd generation languages such as FORTRAN and COBOL. A 4GL should include a dictionary, a data entry facility, a reporting facility, and transaction processing capability. Example 4GLs include PROTOS, SPEEDWARE, and POWERHOUSE, all of which should have the intrinsic ability to perform the actions indicated for relational simulation (multidataset linking and extraction of desired data into subfiles, pref-erably keyed or internally described in some fashion).

DATABASE DESIGN (OR REDESIGN)

Several principles of general database design will be presented in the following paragraphs. Some of these will undoubtedly stick in the craws of designers and analysts since they are fairly radical and generally unheard of, but the author has found them to be valid, and of considerable use in the performance of his design, programming, and maintenance duties.

One thing to remember in the course of reading this paper, time spent in retrieving records, especially in the case of monster datasets (100,000 to 500,000 records and greater), is more important than additional time spent during data entry. A six hour delay in making a decision which must be based on information provided by data retrieved from such large sets by serial read could be very costly; a 2 to 3 second delay in entering an individual record because of the IMAGE–generated DB calls necessary to set the pointers in the key fields, while seemingly unacceptably time–consuming in the aggregate, is of little consequence when considered on a daily basis for a given operator. Rapid access to information has become considerably more important than speed of data entry.

Major principle the first: all databases should be as thoroughly normalized as possible. Datasets should be of the shortest length compatible with efficient operation, and all the data in the given dataset should be closely related. Although the theoretical ideal is one field per dataset, this is not practical in real world applications. However, the number of fields/items in a dataset should be kept as low as possible. Normalization also allows the spreading of datasets individually onto separate disc–drives, which speeds operations considerably. Years ago, storage space was a prime consideration in database design; this is no longer true — with the advent of new drive technologies, data storage has become incredibly inexpensive

(relative to past costs, of course), and no longer need have influence on the shape of our databases.

The use of array fields/items in datasets is usually frowned upon by IMAGE database designers, since QUERY/3000, the original querying facility for the IMAGE database system, was not capable of handling arrays other than indicating the first occurrence of a given array, even though IMAGE has had array capability since its inception. All 4GLs should be capable of handling arrays in a dataset record, so this prohibition no longer has validity; in fact, in most cases, it will make coding of reports and transaction processes much easier if arrays are used in database records. When putting a total for a month into an annual array, it is much faster to be able to reference one occurrence of the array with a variable index than to create a complex IF–THEN–ELSE statement to access the individual items separately by unique name; many other situations would benefit from this precept as well.

Each dataset requires at least one key field to be accessible at all. Additional keys to facilitate record retrieval may be used, but no more than four should be used in an individual dataset. If more than four keys would be required, the dataset is too long in terms of unrelated data items and should be normalized again (broken into two or more new datasets). Sorted chains can be utilized to enforce some sort of order on the records in a dataset for reporting and retrieval purposes, and designers should capitalize upon IMAGE's ability to extend sortfields with proper arrangement of fields within the dataset record. The principle of limiting extension of sort by placing the sort field at the end of the record is still valid, but, in some cases, extension of the sort across five or six additional fields/items may be desirable.

Some ideas for useful key types include zoned fields six bytes in length for date fields, the K2 and K4 numeric items (unsigned integer) for arithmetic keys, SOUNDEX keys to provide an additional retrieval capability on fields containing names, and initials–keys for use in records relating to people.

If 6–byte zoned fields are used for dates, for instance, the structure presented below could provide most of the retrieval possibilities necessary for date–keyed records (the example is in POWERHOUSE, but should be easily translatable to other 4GLs). The date should be stored in YYMMDD order for maximum utility and least programming manipulation.

```
RECORD KEYED-BY-DATE
  ITEM DATA-FIELD-1      CHARACTER SIZE ??
  ITEM DATA-FIELD-2      INTEGER SIZE ??
  ITEM DATE-FIELD        ZONED UNSIGNED SIZE 6
  REDEFINED BY
    ITEM DATE-KEY        ZONED UNSIGNED SIZE 4 &
                         KEY LINKS TO KEY-SET &
                         SORTED ON DAY-FIELD
                         ; YYMM portion of date field
    ITEM DAY-FIELD       ZONED UNSIGNED SIZE 2
    END
  ITEM SORT-EXTENDER     (DESIRED TYPE) SIZE ??
```

K2 and K4 unsigned fields are not recognized or reported by QUERY, but they can be used as indices for sorted chains or as key fields without the problems generated by signed integers. Since most fields of this type deal with positive integers only, the sort comes out in the desired order, not effectively inverted; negative numbers are sorted in reverse order after the positive numbers, since numerical sorts are performed in binary and bit 15 is the sign bit with 1 indicating negative. The problem with QUERY can be effectively ignored, since most reporting will be performed only through the 4GL.

SOUNDEX (SOUNd inDEX) is based on the idea that many letters of the alphabet produce the same general class of sound. Using this idea, names can be grouped by similarity of sound, for instance, Smith, Schmidt, and Smythe. Providing a key field based on SOUNDEX can speed retrieval of records when a particular key is not known exactly; if the last name of a customer could be Johnsson, Johnson, Johnssen, or Johnsen, all possibilities can be more rapidly checked using a SOUNDEX key. POWERHOUSE 5.01 has a SOUNDEX routine built in, and if other 4GLs do not have a similar routine, it should be relatively easy to construct one (appendix A contains the principles used for SOUNDEX creation). Initials can be used for keying personnel records in a similar fashion, although to get proper sorting, they should be constructed as L(astname)F(irstname)M(iddlename). In a personnel management information system recently constructed at the author's company, personnel records are keyed both by employee number and by initials, since the employee number is not as likely to be immediately called to mind when one needs to do casual and relatively rapid information retrieval.

In many cases, juncture files can be used to relate other data-sets. A juncture file is basically two or more key fields linking the same number of datasets. For instance, in a skills resume operation, the juncture file between the personnel file and the skills file contained items for a repeating key on personnel (employee number and initials), a repeating key on skill (by skill code), date of last update, and the number of months experience in the skill, a total of five fields. This design allows an unusual retrieval setup: keyed retrieval by skill, returning into the same dataset keyed by employee number (from the original record retrieved) to get all of each employee's skills. If the skill code retrieved indicates database design, for instance, all other skills for those employees who are database designers are retrieved at the same time, and appropriate resumes can be prepared easily. An example of the ACCESS, SORT, and REPORT statements is shown below. Note that all sorting and reporting references the second file; this file contains the original record retrieved as well as all the others for the employee. This is, of course, just a skeleton of the real report, with only basic information.

```
ACCESS SKILLS-LIST &
        LINK EMP-NO OF SKILLS-LIST &
                TO EMP-NO OF SKILLS-LIST ALIAS OTHER-SKILLS

SORT ON EMP-NO OF OTHER-SKILLS

REPORT &
        EMP-NO OF OTHER-SKILLS &
        SKILL-CODE OF OTHER-SKILLS
```

Major principle the second: databases do not require (and should not have) manual masters. Databases should contain only detail datasets and the automatic masters to which they are keyed. In fact, manual masters tend to slow overall retrievals down. Most manual masters are used for storage of static data: customer data, lookup definitions keyed by codes, etc. Volatile data is generally stored in detail datasets, or should be; this also enables additional keys to be defined for more rapid retrieval, and storage of multiple records for the same key value. Since IMAGE essentially employs a FIFO queue to determine the order in which DB calls are handled, calls to manual masters for lookup clog the system. Lookup or reference data should be accessible, but should not be part of the database proper.

Static data for lookup should be kept in KSAM files outside the database proper. This technique has several advantages. KSAM

handles retrievals for multiple users/sessions at appreciably the same time; calls to KSAM are not shunted into a queue — therefor, retrieval is to all appearances faster. Effectively, although IMAGE DB intrinsics are actually faster on a individual basis than KSAM intrinsics, KSAM is faster in the aggregate. KSAM also allows secondary repeating (non-unique) keys, which IMAGE manual masters do not, and partial-key retrieval, which IMAGE does not. A customer address and information file in KSAM, which will see little change generally, can be keyed additionally to enable retrieval of all clients in a given state, all with a given zip–code (and a real bonus, all with the same first three digits of a zip–code), etc. It should be noted that KSAM is not capable of performing its partial–key retrievals on most numeric data; to enable partial key calls on number sequences, the field must be zoned, and leading zeros included in the partial–key value being used.

There are several standard complaints about KSAM. Answers hopefully satisfactory are provided for the three most common.

No logging in KSAM: Who needs it? This is static data we're talking about — it doesn't change frequently, and the last fulldump should enable reloading of the file with a minimum of fuss, and virtually no loss of integrity.

Files can be corrupted or destroyed by a crash: This is partly answered in the previous complaint — reloads are very easy. KSAM files are destroyed if being written to during a crash, not just open for reading — the static data answer applies again. Additionally, HP3000 machines and MPE are very reliable, and crashes are becoming less and less frequent.

KSAM is too hard to work with, the intrinsics are all different: Programmers and database designers will not be working with KSAM *per se*. 4GLs handle all of the dog work connected with access to both KSAM and IMAGE (and if yours doesn't, it might be time to investigate others). When was the last time you used a DB intrinsic call if you're generally working with 4GLs?

A set diagram of a database constructed using the principles described so far may be found on as Appendix B. This is the football pool database mentioned previously. The actual database and attendant programs and documentation (in POWERHOUSE source code) may be found on the swap tape for this conference; a general set of operating rules for the pool is given in Appendix C..

Major principle the third: don't overload automatic masters with too many paths. It is a temptation to cram as many paths as possible into an automatic master, and the limit is quite high — 15 paths are allowed by IMAGE. But just as many paths into a detail dataset degrades performance in accessing and updating the detail, too many paths into an automatic master also causes problems. It should make no difference at all, but more than nine paths from an automatic master to various detail datasets seems to cause an almost asymptotic drop in performance. While there is a fairly high limit (63) on the number of datasets that an IMAGE database may contain, it is better to keep that number fairly low as well. Database normalization should include the principle that only related datasets should be in a given database. 4GLs have no problems handling multiple databases, and a larger number of smaller databases could contribute to a small speed up in access times, since a retrieval could be spread across several databases rather than one and would avoid the problem with stacking in the IMAGE "queue".

A suggested method for avoiding this possible problem is to trace no more than one path to an automatic master from an individual detail dataset, and to group paths by class of data rather than strictly by type of item. All the year–month key paths for completion, for instance, could be traced to one automatic master, while the year–month key paths for starting date could be traced to another. There is seldom more than one such key per detail dataset (at least there shouldn't if the database is well designed). So this method should resolve the problem before it appears.

Another general principle: transaction processors tend to work faster than reporters. Use a transaction process to extract the data into a subfile, linking as many datasets as necessary to provide all the data needed, then run the report from the subfile. The transaction process will run fairly swiftly since it accesses those additional records which it is using by key, even on a serial read on the base dataset; if the base file for the process can be read by key, it will be even faster, since a good portion of the records in the base dataset will be effectively skipped or ignored (how about never even accessed!). Select only those data fields needed for the report when you specify the subfile, but do include lookup data that might come from very large records, particularly if it is likely to be used as a sort value. This process will form an extract table; in many cases, this extract table may be used for several different reports if it is thoughtfully built — but don't put too much information into it. The

smaller the record size, the more records that will be in a block, and the faster the various programs accessing the table can run, especially if sorting is involved. Sorting will be considerably easier for the system to accomplish if the record size in the extract table is relatively small, since the size of any sortfile (which requires space on the disc) will be determined by the size of the subfile being used, plus the size of any records being added for lookup. As an example of the time saving available using this method, a report run directly from the database, with a number of arithmetic and accumulation operations, required over sixteen hours to run to completion; the same report ran in three minutes when prepared from an extract file selected by a transaction process which required an hour and a quarter to run. This shows considerable time saving — a 12 to 1 differential on the runtime for this particular report. Even with the time figured in for writing the extraction process (about five hours), there was still a 2 to 1 difference in time spent.

When creating extract tables, remember that a field being used as a key in a dataset which is the object of a link does not necessarily have to be a key in the calling dataset (the subject of the link). Any field can provide a value, of any type, to be used for the link; various conversion utilities are available in the 4GLs to facilitate data conversion from one form to another. The only requirement in this respect is that the linking field or value must be of the same general type (character, integer, zoned, etc.) as the key to which it is being linked, and that it must be neither longer nor larger that the key being called (40 characters linking to 20 "jest don't work", although 20 linking to 40 will; a value greater than 65535 or less than 0 can't be used to link to a K2 key; etc.).

SIMULATION OF RELATIONAL STYLE

To be able to simulate a relational style with an IMAGE database, several items are needed: a well designed database, a 4GL, and a database handler.

The well designed database may be constructed (or reconstructed, as the case may be) using the principles outlined above. Database designers should strive to have their products well designed whether they are using this methodology or not — it cuts down on maintenance. Many 4GLs are available on the market for quite reasonable prices, and no site should be without one (not if the site expects to keep up with the Joneses, so to speak). The main purpose

of the DBH is to allow path changing as necessary, which happens to be a major component of the simulation.

Since data can be retrieved from detail datasets quickly only if the search item is a key (or if the dataset has comparably few records in it), being able to add and/or delete paths into datasets is a virtual *sine qua non* of the simulation process. Granted, such a change to a database requires a good deal of time, since both the automatic master and the detail set must be rebuilt by the DBH to provide room for the new pointers for the additional path. However, assuming the database was fairly well designed and constructed in the first place, this restructuring should not be a daily occurrence. In many cases, the process can be completed overnight by using a stream job, or over a weekend, or by someone calling into the system by modem from home (most of the people involved in database changes of this caliber seem to have terminals and modems at home, and beepers with which the operations people on site can scream for help if necessary). DBGENERAL has a batch facility already, and it is well documented, and the line of answers for ADAGER can be traced rather easily; if the process is done frequently, permanent job streams can be set up which require only changes of the names to function properly. Whatever method is used, it is highly recommended that a backup of the old version of the database be performed prior to the change process, and a similar backup of the newly redesigned database be made as soon as the change process is completed; Murphy operates in any situation in which he can!

To recap: simulation of relational database style can be accomplished using IMAGE by arranging (and/or rearranging, using a DBH) the database to allow the swiftest retrieval of pertinent data, which is put into extract tables using a 4GL, and then reported from the extract table by a 4GL report writer. This may seem rather simplistic, but that's about all there actually is to it.

Appendix A

SOUNDEX Principles

SOUNDEX routines should be constructed to use the following principles, paraphrased from the POWERHOUSE manuals.

1. Always keep the first character in the word as a character, whether vowel or consonent.

2. Remove all but the first instance of adjacent identical letters.

3. Drop all vowels (A, E, I, O, U) including W, Y, and H, unless the vowel is the first letter of the word.

4. Replace all letters but the first with the following numeric values:
   - 1 for letter B, F, P, or V
   - 2 for letter C, G, J, K, Q, S, X, or Z
   - 3 for letter D or T
   - 4 for letter L
   - 5 for letter M or N
   - 6 for letter R

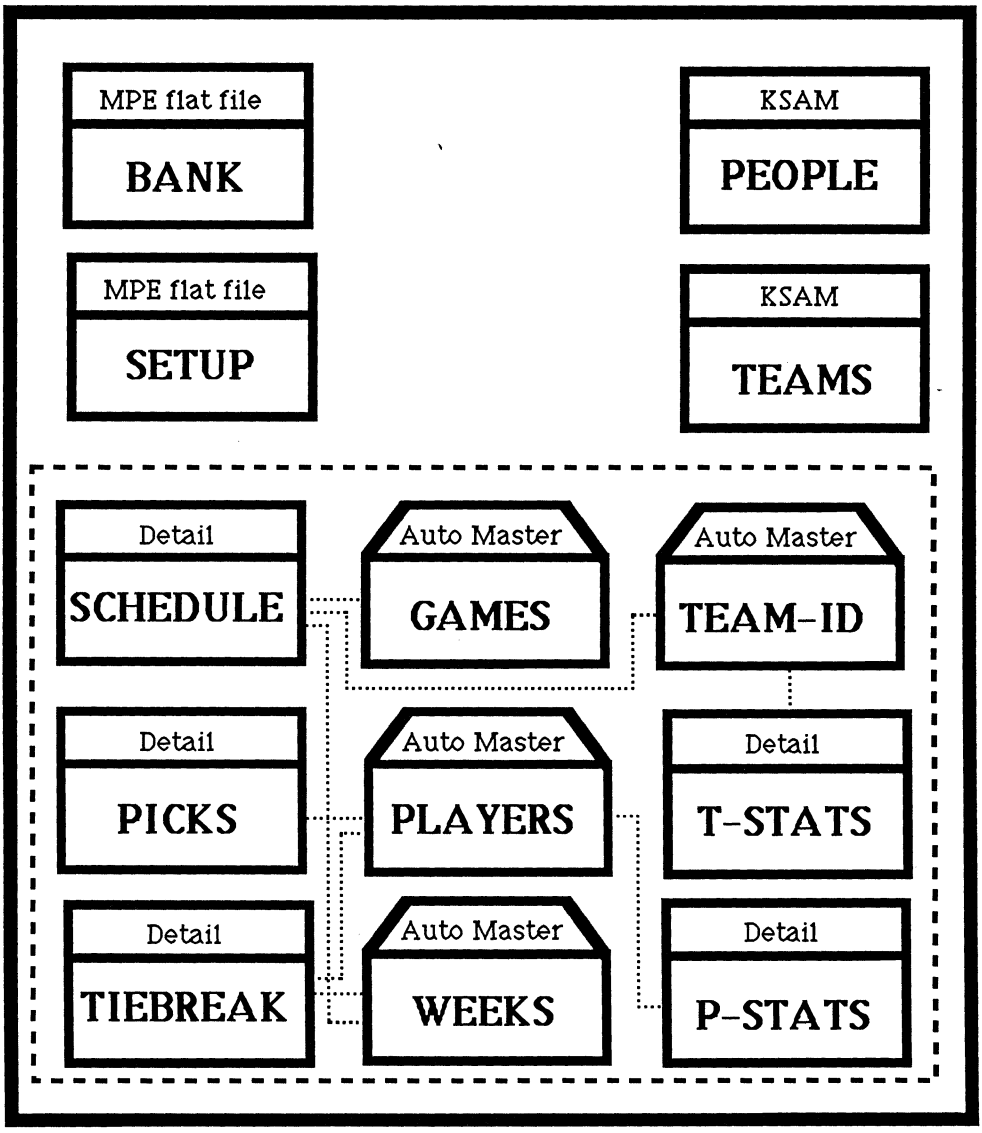
5. If adjacent assigned numeric values are equivalent, remove all but the first instance.

6. Pad the remainder of the result with zeros.

The rules just provided will handle most words or names in English. Other languages can not be guaranteed for proper results.

Appendix B

SET DIAGRAM FOR THE FOOTBALL POOL DATABASE



Appendix C

Operating the Football Pool

The football pool database provided on the CSL as an example of the principles expounded in the foregoing paper is fairly easy to operate. All the data entry screens, transaction processes, and reports required are included on the tape. These are provided in source form only, and must be compiled to function. The account structure for the pool overall is designed for three groups: **.DATA**, **.SRC**, and **.X**. All of the source files have file type codes to describe their function, with a given source having a code 100 lower than the corresponding compiled POWERHOUSE file code: 540 is dictionary source, 541 is QUICK screen source, 542 for QUIZ reports, and 543 for QTP runs. If subfiles are built, the name begins SF; most subfiles are temporary, but there are some permanent ones — there are, however, SIZEd to a limit of 500; if there are more players than expected, raise the limits.

Make whatever changes might be necessary in terms of group and account to fit the pool onto your system. The source uses POWERHOUSE 5.01. Compile the dictionary after making appropriate changes, then CREATE the database and auxiliary KSAM and MPE files. Three MPEX 'user' template files are provided in **.SRC** to allow easier compilation of the files in the system. Several documentation files are provided in **.SRC** (code 549) to provide more in—depth explanations.

The first and one of the most important aspects of operating the pool is to enter the team names and the season's schedule on the appropriate screen. Then decide what the amount of each entry will be ($2 to $3 is suggested), and whether a portion will be reserved for end of season payout (best and worst averages, season high and low scores, most wins, highest and lowest total points scored, etc.) — if so, how much (as a percentage). The pool's programs will calculate how much is to be paid to the winner of the pool each week, depending on the entry fee, less any reserved amount, and the number of entries for a given week; it also keeps track of the reserved amount due at season's end. The minimum number of weeks played in may also be set at this time to determine winners' eligibility for end of season bonuses.

Playing the pool is based on the line established for each week's games; this should be picked up on Tuesday morning. The spread

(the points given to a team to ensure an even bet in each game) is included for all scoring. If the Giants beat the Bears by 10 (a touchdown and a field goal), but the line gave the Bears a 13-point spread (they had to be beaten by two touchdowns), then for purposes of the pool, the Giants have lost and the Bears have won. There are fourteen games played each week; players select the team they expect to win in each match-up (spreads included), and assign a "confidence factor" of from 1 to 14 to that game (each number is used only once) — a player's final score each week is the total of the confidence factors for the games in which s/he has selected the winning teams. The pick sheet should be returned to the pool co–ordinator by 6:00pm Thursday evening (there are occasionally Thursday games). Winners can be announced the following Monday morning, unless a tie-breaker is necessary, in which case the Monday night game should suffice (although a total points guess on Monday night's score serves as a further tie-break).

Players are entered into the system only the first time they play. After that, the code assigned to each player is used to keep track of various situations, including entries and statistics; a player may also enter more than once in a week — extra codes are generated for additional entries by the same player. If the base code for a player is LKD01A (Kerry David Lloyd, first entry for the week, first set of initials LKD), then the second entry in a week is LKD02A, and the third LKD03A. The programs also check for like initials, and will assign B to the second player with the same initials (LKD01B for Kevin Donald Lucas), C to the third, etc. Statistics are kept for both teams and players, with printouts available for team records (actual won/lost, times spread beaten, total points scored, total points scored by opponents), and each 4 weeks (or bi—weekly or weekly if desired) for player standings, divided into sections according to number of times played.

Printouts available include: the following week–end's games with spreads, scores for the previous week with winning players and amounts awarded, weekly team statistics (as described above), and player statistics. Screens are designed to allow the easiest and fastest data entry; overall time to run the pool can be as little as 2 hours a week for 50 to 60 players.

ELIMINATE THE 4 AM BLAHS !

By *Pat Lockwood* & *Joe Junker*

Orion Systems Technology, Inc.
6534 Place D'Valencia
Phoenix, AZ 85014
(602) 840-9157

INTRODUCTION

How many times have you received a phone call like one of the following, particularly in the middle of the night?

> Joe, you know that job that adds FRIVITS transactions to the ARPGNU data base? Well, it just blew up with an IMAGE error number 16. Will you come in and do something.. this job has to finish by morning.

or Hey, Pat, we were running the LVRWURST job and it collapsed! When Herman was the programmer on this job, he used to come in and change some file equations, but I don't know exactly how. Help me, will you. The boss says this job is hot.

Most of these can be avoided; MPE's file system, the IMAGE intrinsics, and a little planning can help you eliminate those 4 AM BLAHs.

This paper will explore the use of some standardized techniques that can easily be used by COBOL programmers (and others) to predict potential problems, and then take corrective action before they occur.

The specific problem addressed here is the lack of space in a file or data base for holding transactions to be posted. That sounds pretty basic, but historically has been a serious problem for data processing operations departments. This seems to be particularly true for purchased applications, and shops that have converted to the HP3000 from other systems. It's also true in new development in shops that create long job streams with many builds of new files.

To determine how to prevent the problem, let's first examine some situations that cause it.

Unless *totally* on-line, most systems have some combinations of the following posting type programs:

DB to DB	Typically used for conversions and inter system data feeds.
DB to File	Used for extracts, sorts, etc.
File to DB	Commonly used for transaction posting, conversions, and inter system interfaces.
File to File	Used for rebuilds of KSAM files, transaction to master updates in older batch oriented systems, and other similar work.

All of these share a common characteristic; if you know your input volume, you can predict the amount of space required for your output. This means that you can either create enough space for the target, or ensure that enough exists before starting the update process.

And, if you can do that, you will save many headaches and minutes (or hours) required to restore the files or data base before restarting the application with, what you hope is, the appropriate space available.

Let's look at some samples of the simple rule, "Know your input volume and build or verify the required output space".

DATA BASE to DATA BASE

The first example is fairly simple; read a data set from one data base, and write it into another. For the sake of the example, we'll assume we're reading the INPUT data set of OLDDB, and writing to the OUTPUT data set of NEWDB. Further we'll assume that all of INPUT must be written, and that OUTPUT is empty. (This is a conversion job).

The following page contains the IMAGE calling parameters we'll be using.

```
01 DB-BASE-NAME.

     05 DB-OPEN-NUM              PIC X(2).
     05 DB-OPEN-FILE             PIC X(24).

01 DB-STATUS.

   05 DB-CONDTN-WORD        COMP PIC S9(4).

      88 DB-OK                          VALUE ZERO.
      88 DB-NOT-FOUND                   VALUE 17.

   05 DB-STAT-REC-LGTH      COMP PIC S9(4).
   05 DB-STAT-REC-NUM       COMP PIC S9(9).
   05 DB-STAT-NUM-ENTRIES   COMP PIC S9(9).
   05 DB-STAT-PRIOR-REC-NUM COMP PIC S9(9).
   05 DB-STAT-BEGIN-CHAIN   COMP PIC S9(9).

01 DB-SET-NAME                   PIC  X(16)  VALUE SPACE.

01 DB-MODE1                 COMP PIC S9(4)    VALUE 1.
01 DB-MODE202               COMP PIC S9(4)    VALUE 202.

01 DB-INFO-DATA.

 02 DB-202-NO-ENTRIES-AVAIL COMP PIC S9(9).

 02 DB-INFO-BUFFER-202.

   05 DB-202-NAME                 PIC  X(16).
   05 DB-202-TYPE                 PIC  X(1).
   05 FILLER                      PIC  X(1).
   05 DB-202-WORD-LENGTH     COMP PIC S9(4).
   05 DB-202-BLOCKFACTOR     COMP PIC S9(4).
   05 FILLER                 COMP PIC S9(4).
   05 FILLER                 COMP PIC S9(4).
   05 DB-202-NO-ENTRIES      COMP PIC S9(9).
   05 DB-202-CAPACITY        COMP PIC S9(9).
```

In addition to the standard IMAGE parameters, we have DB-INFO-DATA; a most handy little buffer. It's used for the return of information from a call to DBINFO, mode 202.

This is shown on the following page.

Remember, we need to know how many entries there are in the INPUT data set, and if there's enough room to write them to the OUTPUT data set. The IMAGE intrinsic DBINFO can determine both. This example assumes an empty data set in OUTPUT for a conversion program.

```
MOVE OLDDB base name          TO DB-BASE-NAME.
MOVE "INPUT"                  TO DB-SET-NAME.

CALL "DBINFO"              USING DB-BASE-NAME,
                                 DB-SET-NAME,
                                 DB-MODE-202,
                                 DB-STATUS,
                                 DB-INFO-BUFFER-202.

IF ( NOT DB-OK ),

    PERFORM db-error-routine

    GOBACK.

MOVE DB-202-NO-ENTRIES        TO input-count.
```

Assuming we had no error conditions, we now have a count of the number of entries in the INPUT data set of OLDDB stored in *input-count*.

All we need to know now is the amount of space available in the OUTPUT set of NEWDB. We can use the same intrinsic.

```
MOVE NEWDB base name          TO DB-BASE-NAME.
MOVE "OUTPUT"                 TO DB-SET-NAME.

CALL "DBINFO"              USING DB-BASE-NAME,
                                 DB-SET-NAME,
                                 DB-MODE-202,
                                 DB-STATUS,
                                 DB-INFO-BUFFER-202.

IF ( NOT DB-OK ),

    PERFORM db-error-routine

    GOBACK.

IF DB-202-CAPACITY < input-count,

   PERFORM data-set-full-error-routine

   GOBACK.
```

If there isn't enough room (DB-202-CAPACITY) to hold the input records (*input-count*), then there isn't any reason to continue processing. We can expand the OUTPUT data set of NEWDB without having to first restore or erase it due to having it partially updated.

DATA BASE to FILE

The previous example was a fairly simple case; loading a data set into an empty data set that already exists. It's fairly common to need to know how big a file must be to hold entries to be extracted from a data set. Of course, if you use a BUILD command in a permanent job stream, you're stuck with it.

A more reliable way is to build the file from within your program to the size desired, based upon the number of records to be extracted.

The next example will be a little more complex. In this case, we're extracting only a subset of the INPUT data set. It was built with a search item, YR-MO. The rules for the extract are:

> Select only those records that are between START-YR-MO and END-YR-MO, inclusive. The extracted records are to be written to a file for later processing.

We want to ensure that the file is large enough to hold all of the extracted records, but certainly don't want to build it to some excessive size that will waste disc space.

We'll need a little more working storage for this one. The next page illustrates the additional work areas used.

```
01 BUILD-COMMAND.

   05 FILLER                    PIC X(38)  VALUE
      "BUILD outfile;REC=-170,24,F,ASCII;DISC=".
   05 NUM-RECS                  PIC 9(9)   VALUE ZERO.

01 COMMAND-LINE.

   05 MPE-COMMAND               PIC X(79).
   05 FILLER                    PIC X(1)    VALUE %15

01 COMMAND-ERROR          COMP PIC S9(4).
01 COMMAND-PARAM          COMP PIC S9(4)

01 START-YR-MO.

   05 START-YR                  PIC 9(2).
   05 START-MO                  PIC 9(2).

01 END-YR-MO.

   05 END-YR                    PIC 9(2).
   05 END-MO                    PIC 9(2).

01 FIND-YR-MO.

   05 FIND-YR                   PIC 9(2).
   05 FIND-MO                   PIC 9(2).
```

Let's take a look at another way of determining how many records we may have to process. Since we only need a subset of the INPUT data set, using DBINFO might save us from disaster, but would also tell us the total universe from which we only want a portion.

DBFIND does more than locate the first entry in a chain; it also tells us how many entries there are. And that can be a valuable piece of information.

On the next two pages, we'll look at a short section of code that uses this facility.

```
FIND-AND-COUNT-INPUT.

   MOVE "INPUT"                  TO DB-SET-NAME.
   MOVE "YR-MO"                  TO DB-ITEM-NAME.
   MOVE START-YR-MO              TO FIND-YR-MO.

   PERFORM ADD-TO-NUM-RECS.

   IF NUM-RECS = ZERO,

      PERFORM nothing-found-routine

      GOBACK.

   MOVE "PURGE outfile"          TO MPE-COMMAND.

   CALL INTRINSIC "COMMAND" USING COMMAND-LINE,
                                  COMMAND-ERROR,
                                  COMMAND-PARAM.

   MOVE BUILD-COMMAND            TO MPE-COMMAND.

   CALL INTRINSIC "COMMAND" USING COMMAND-LINE,
                                  COMMAND-ERROR,
                                  COMMAND-PARAM.

   IF COMMAND-ERROR NOT = ZERO,

      PERFORM command-error-routine

      GOBACK.
```

Continued ....

```
ADD-TO-NUM-RECS.

    CALL "DBFIND"              USING DB-BASE-NAME,
                                     DB-SET-NAME,
                                     DB-MODE1,
                                     DB-STATUS,
                                     DB-ITEM-NAME,
                                     FIND-YR-MO.

    IF ( NOT DB-OK ) AND
       ( NOT DB-NOT-FOUND )

        PERFORM db-error-routine

        GOBACK.

    ADD DB-STAT-NUM-ENTRIES      TO NUM-RECS.

    IF FIND-YR-MO < STOP-YR-MO,

       IF FIND-MO = 12,

          MOVE 1                 TO FIND-MO
          ADD  1                 TO FIND-YR

          GO TO ADD-TO-NUM-RECS

       ELSE,

          ADD  1                 TO FIND-MO

          GO TO ADD-TO-NUM-RECS.
```

A series of DBFINDs allows us to add the number of entries found before we issue the build command. The paragraph FIND-AND-COUNT-INPUT sets up, and when all input is counted, it issues a BUILD command to create the *outfile* to the desired size.

The paragraph ADD-TO-NUM-RECS is repeated for each YR-MO desired, adding the number of entries found directly into the NUM-RECS field in the BUILD command.

FILE to DATA BASE

Now things get a little more complex. This example requires reading a file, and posting the records to a data base. The data set to which we're posting isn't empty, so we need to know how many records are in the file, and how much room is left in the data set before we start the posting program.

Lucky for us, MPE's file system has an intrinsic we can use to determine the number of records there are in a file; it's FFILEINFO. To use it, we'll again need some work areas.

```
01 FILE-NAME                PIC X(36).
01 FILE-NUM           COMP PIC S9(04).
01 F-OPTIONS          COMP PIC S9(04).
01 A-OPTIONS          COMP PIC S9(04).
01 ITEM-NUM           COMP PIC S9(04).
01 FSERR-NUM          COMP PIC S9(04).
01 REC-COUNT          COMP PIC S9(09).
01 FSERR-MSG                PIC X(72).
01 FSERR-MSG-LENGTH   COMP PIC S9(04).
```

So, for this example, we'll use FFILEINFO to determine our input volume, and then use DBINFO to see if we have enough space in the data set to post the file. This sequence is fairly common in conversions, inter system communications, and other applications that use files of transactions that are later posted to a data base.

```
MOVE infile              TO FILE-NAME.
MOVE ZERO                TO FILE-NUM,
                            FSERR-NUM,
                            REC-COUNT.
MOVE %7                  TO F-OPTIONS.
MOVE %2300               TO A-OPTIONS.
MOVE 10                  TO ITEM-NUM.
```

F-OPTIONS are set to %7 (bit pattern 0 000 000 000 000 111); this will be used by a call to FOPEN looking for an ASCII file, first permanent, and if not found, then look for a TEMP file.

A-OPTIONS are set to %2300 (bit pattern 0 000 010 011 000 000); to be opened for SHARE, GMULTI access.

The 10 used in ITEM-NUM will be used to tell the FFILEINFO intrinsic that we want the tenth item, which is the location of the file's EOF which really means its record count.

The sequence of instructions to get the input volume will be:

- Open the file with FOPEN
- If unable to open it, use FCHECK to get the FSERR-NUM and then use that to get the FSERR error message and exit
- Get the record count using FFILEINFO
- Close the file using FCLOSE.

The next page shows the COBOL code that will accomplish these.

```
CALL INTRINSIC "FOPEN"       USING FILE-NAME,
                                   F-OPTIONS,
                                   A-OPTIONS,
                                   \\, \\, \\, \\,
                                   \\, \\, \\, \\
                             GIVING FILE-NUM.

IF FILE-NUM = ZERO,

   CALL INTRINSIC "FCHECK"   USING FILE-NUM,
                                   FSERR-NUM,
                                   \\, \\, \\

   MOVE 72                    TO FSERR-MSG-LENGTH

   CALL INTRINSIC "FERRMSG" USING FSERR-NUM,
                                   FSERR-MSG,
                                   FSERR-MSG-LENGTH

   PERFORM file-error-routine

   GOBACK.

CALL INTRINSIC "FFILEINFO"   USING FILE-NUM,
                                   ITEM-NUM,
                                   REC-COUNT.

CALL INTRINSIC "FCLOSE"      USING FILE-NUM,
                                   0, 0.
```

The backslashes are used in COBOL to indicate optional parameters that are not being passed.

If unable to open the file, FOPEN leaves a ZERO in FILE-NUM. Therefore we can easily check that to see if there was some type of error attempting to open it. FCHECK returns the FSERR-NUM (file system error number) for the last attempted open if the FILE-NUM is ZERO.

On the next page we'll use DBINFO to determine how much space is left in the OUTPUT data set.

```
MOVE "OUTPUT"                TO DB-SET-NAME.

CALL "DBINFO"              USING DB-BASE-NAME,
                                 DB-SET-NAME,
                                 DB-MODE-202,
                                 DB-STATUS,
                                 DB-INFO-BUFFER-202.

IF ( NOT DB-OK ),

    PERFORM db-error-routine

    GOBACK.

COMPUTE DB-202-NO-ENTRIES-AVAIL

      = DB-202-CAPACITY
      - DB-202-NO-ENTRIES.

IF DB-202-NO-ENTRIES-AVAIL < REC-COUNT,

   PERFORM data-set-full-error-routine

   GOBACK.
```

This time we added a little to our DBINFO routine. Since DBINFO returns both capacity and number of entries, it's very easy to determine the amount of space still available.

So we're still following the basic rule of knowing input volume to build or verify required output space.

We've seen how to determine the number of input records in a file or data base, and how to determine space available in a data set. Sometimes you're reading one file and creating another one, as in building a KSAM file. The same principles apply to this situation.

We used FFILEINFO to get an input record count in the previous example. FFILEINFO is a fairly new intrinsic, and is very flexible. There's an older intrinsic, FGETINFO, that is very similar. It's not quite as flexible, but it does allow you to obtain more information about a file in one call than FFILEINFO, which is limited to only five items of information per call.

For work areas, we'll use the following.

```
01 GETINFO-FILENUM          COMP PIC S9(04).
01 GETINFO-FILENAME              PIC X(28).
01 GETINFO-FOPTIONS          COMP PIC S9(04).
01 GETINFO-AOPTIONS          COMP PIC S9(04).
01 GETINFO-RECSIZE           COMP PIC S9(04).
01 GETINFO-DEVTYPE           COMP PIC S9(04).
01 GETINFO-LDNUM             COMP PIC S9(04).
01 GETINFO-FILECODE          COMP PIC S9(04).
01 GETINFO-RECPT             COMP PIC S9(09).
01 GETINFO-EOF               COMP PIC S9(09).
01 GETINFO-FLIMIT            COMP PIC S9(09).
01 GETINFO-LOGCOUNT          COMP PIC S9(09).
01 GETINFO-PHYSCOUNT         COMP PIC S9(09).
01 GETINFO-BLKSIZE           COMP PIC S9(04).
01 GETINFO-EXTSIZE           COMP PIC S9(04).
01 GETINFO-NUMEXTENT         COMP PIC S9(04).
01 GETINFO-USERLABELS        COMP PIC S9(04).
01 GETINFO-CREATORID              PIC X(08).
01 GETINFO-LABADDR           COMP PIC S9(09).
```

As you can see, FGETINFO returns a tremendous amount of information about a file, and you can get it all with only one call.

We'll also use a little different technique to create our new KSAM file for this example. When an output file is opened in COBOL, a new TEMP file is created if none exists, and its characteristics can be established by a file equation issued before it is opened.

So, we'll create a special file equation for this.

```
 01 KSAM-FILE-EQUATE.

    05 FILLER                    PIC X(46)   VALUE
       "FILE outfile;REC=-512,8,F,ASCII;DEV=,,20;DISC=".

    05 KSAM-NUM-RECS             PIC 9(09)   VALUE ZERO.
```

This is almost like the BUILD command we used earlier in the DATA BASE to FILE example, except we've added a little performance trick. The "DEV=,,20" parameter is used when loading a new KSAM file. This tells the file system to allocate twenty key block buffers in the KSAM extra data segment, which can significantly decrease disc access, and therefore throughput time, when loading a new KSAM file.

This, combined with file blocking to 4096 bytes, makes writing the new file a pretty fast operation. The 4096 limit on blocking allows us to write fairly large blocks without exceeding the maximum size that can be handled efficiently by the newer "XP" disc drives with "controller cache".

Since it can return so much information in one call, it's pretty easy to set up standard work areas and a commonly used routine to call FGETINFO. In COBOL, this might be in your COPYLIB, or you can use EDITOR "JOIN" files for other languages.

Calling FGETINFO is like calling FFILEINFO; first you call FOPEN, check for errors, then call FGETINFO and FCLOSE the file. So for our example, we'll skip those steps, and get right to the call to FGETINFO on the next page.

```
CALL INTRINSIC "FGETINFO"   USING GETINFO-FILENUM,
                                  GETINFO-FILENAME,
                                  GETINFO-FOPTIONS,
                                  GETINFO-AOPTIONS,
                                  GETINFO-RECSIZE,
                                  GETINFO-DEVTYPE,
                                  GETINFO-LDNUM,
                                  \\,
                                  GETINFO-FILECODE,
                                  GETINFO-RECPT,
                                  GETINFO-EOF,
                                  GETINFO-FLIMIT,
                                  GETINFO-LOGCOUNT,
                                  GETINFO-PHYSCOUNT,
                                  GETINFO-BLKSIZE,
                                  GETINFO-EXTSIZE,
                                  GETINFO-NUMEXTENTS,
                                  GETINFO-USERLABELS,
                                  GETINFO-CREATORID,
                                  GETINFO-LABADDR.

CALL INTRINSIC "FCLOSE"     USING GETINFO-FILENUM,
                                  0, 0.
```

The parameter that wasn't requested is one that returns the hardware address of the device. We normally don't request that because it can cause strange results on the newer big HP3000s (Series 64, 68, and 70).

Just as in the previous examples, we now know our input volume; this time it's in GETINFO-EOF. All that remains is to get it into our file equation and open the file.

The code for this is on the next page.

```
   MOVE GETINFO-EOF              TO KSAM-NUM-RECS.

   MOVE "PURGE outfile"          TO MPE-COMMAND.

   CALL INTRINSIC "COMMAND"   USING COMMAND-LINE,
                                    COMMAND-ERROR,
                                    COMMAND-PARAM.

   MOVE "PURGE outfileK"         TO MPE-COMMAND.

   CALL INTRINSIC "COMMAND"   USING COMMAND-LINE,
                                    COMMAND-ERROR,
                                    COMMAND-PARAM.

   MOVE KSAM-FILE-EQUATE         TO MPE-COMMAND.

   CALL INTRINSIC "COMMAND"   USING COMMAND-LINE,
                                    COMMAND-ERROR,
                                    COMMAND-PARAM.

   IF COMMAND-ERROR NOT = ZERO,

      PERFORM command-error-routine

      GOBACK.

   OPEN OUTPUT outfile.

   OPEN INPUT infile.
```

We're now ready to go. Of course, we'll remember to SAVE the KSAM file and its key file after we close the output.

If we were adding to an existing KSAM file, we could have calculated the amount of space remaining in the file using:

```
        COMPUTE room-left

              = GETINFO-FLIMIT
              - GETINFO-EOF.
```

So, in all of our examples we've determined our input volume to build or verify required output space. Of course, sometimes a little more logic is required in your program. Your output will not necessarily have a "one for one" match with your input; but a little imagination and some investigation will normally give you a formula that will work.

These tips work very well for files, and can save restore time for data bases. However, there's still one ingredient missing; that is to not get into the situation where a data base is too small for expected additions during the normal processing cycle.

The answer to this is fairly easy. We've seen how DBINFO can tell you the number of entries and capacity for each data set within a data base. With this knowledge, it's not very difficult to design a program to verify and report on all data sets in a data base on a periodic basis. A simple report to the system manager or data base administrator can point out potential trouble well before it occurs.

The sample report on the next page is one used in our shop. The program is designed to report on any data base, and is run daily for every production data base. It prints the report for a data base only if the percent full exceeds a predetermined rate, which is set low enough to allow data base expansion before the need for space becomes critical.

Now, unplug your telephone, and sleep soundly.

Pat Lockwood & Joe Junker

Eliminate the 4 AM BLAHS!

19

Report: DBCKLIST - 1
DBCHKERX A.01

S O U T H W E S T F O R E S T I N D U S T R I E S
Clarksville Sheet Plant
DATA BASE ANALYSIS for CONSAL

DEC 19, 1986
FRI 7:28 AM

SET NAME	TYPE	CAPACITY	RECORDS	AVAILABLE	% FULL	STATUS	CHANGES
LOCATION-CTL	MASTER	1	1	0	100.00	OK	
EDIT-PARMS	MASTER	53	32	21	60.38	OK	
SALESMEN	MASTER	37	5	32	13.51	OK	
TAX-DATA	MASTER	37	2	35	5.41	OK	
YR-MO-INDEX	AUTO MSTR	41	25	16	60.98	OK	
INVOICE-INDEX	AUTO MSTR	12,007	9,817	2,190	81.76	PURGE	
INVOICE-DETAIL	DETAIL	18,000	15,351	2,649	85.28	CRITICAL	

YOU MEAN YOU DON'T USE HP'S EDITOR?

by

Patricia Lowers and Gene Harmon
AH Computer Services, Inc.
8210 Terrace Drive
El Cerrito, CA USA 94530-3059

INTRODUCTION

THIS MATERIAL IS DIRECTED PARTICULARLY TOWARD THOSE HP-3000 USERS WHO DO NOT HAVE THE ADVANTAGE OF HAVING AN ADVANCED EDITING MULTI-FUNCTION SOFTWARE PACKAGE SUCH AS QEDIT OR MPEX.

THE SUBJECT MATERIAL SHOULD ALSO BE OF INTEREST TO THOSE WHOSE JOB DESCRIPTIONS DICTATE THAT THEY WANDER FROM INSTALLATION TO INSTALLATION, NEVER KNOWING WHAT THEY MIGHT FIND....IS TRUSTY QEDIT OR MPEX THERE TO DO THEIR BIDDING AT 2 IN THE MORNING SO THAT THEY MAY QUICKLY EDIT AND UPDATE THOSE 200 SOURCE PROGRAMS WHICH MUST BE IN PRODUCTION IN JUST 6 HOURS?

SEVERAL TECHNIQUES AND SOME INSIGHTS WHICH MAY HAVE ESCAPED THE CASUAL PERUSER OF THE HP EDIT-3000 MANUAL WILL BE HEREIN PRESENTED. THE MAIN TOPICS WILL BE:

1) WHILE LOOPS
2) USE FILES
3) HOLD FILES
4) QUOTIDIAN CONSIDERATIONS

I. WHY DID I EVER BOTHER WITH THIS AND THE PARTICULAR LEVERAGES THE TECHNIQUES AFFORDED ME?

MOST OF MY TIME IS SPENT IN THE CREATION AND MANIPULATION OF MEDIUM TO LARGE-SIZE (OR, FOR THE MORE TECHNICALLY MINDED: LILLIPUTIAN TO BROBDINGAGIAN) DATABASES ALONG WITH THEIR PROGRAMS. ALTHOUGH THE INSTANCES ARE SO SELDOM AS TO ESCAPE MEMORY, IT SOMETIMES HAPPENS THAT THE WRONG THING HAS BEEN DONE, THAT THE RIGHT THING WAS NOT DONE, OR THE THING NEEDS TO BE DONE IN ANOTHER PLACE . . . AND RIGHT NOW.

WHAT IS NEEDED IN THESE CIRCUMSTANCES ARE WAYS TO CUT DOWN KEY-STROKES, SPEED SEARCHES, ACCOMPLISH GENERIC FILE MANIPULATIONS WITHOUT TOTAL DISASTER (IT MAY BE THE WRONG SIDE OF 12AM, AND THE FINGER-BRAIN-EYES FBE CONSTANT (FBE = BLOOD-SUGAR-LEVEL/GREENWICH MERIDIAN TIME ADJUSTED LOCALLY) IS SIGNALLING MEDIC ALERT); AND GENERALLY GET ONE TO THE CHURCH ON TIME.

MY DEVELOPMENT SOURCE FOR MANY OF THE TECHNIQUES WERE THE BACK PAGES OF THE HP EDIT-3000 MANUAL. THOUGH THE EXAMPLES

GIVEN IN THOSE PAGES WERE A CLEAR ENCOURAGEMENT TO DEVELOP ONE'S OWN WORD-PROCESSOR USING HP EDITOR, THE TECHNIQUES PRESENTED IN THIS PAPER WILL FALL SHORT OF THAT GOAL . . . THAT WILL BE LEFT TO OTHERS.

II. WHAT COMES NEXT?

EMPLOYING THE CASEBOOK METHOD, EACH TOPIC WILL BE APPROACHED AS FOLLOWS:

1. WHAT I'VE GOT. (NOW SADLY FAMILIAR WORDS TO THOSE FEW REMAINING HABITUES OF SINGLES BARS)

   - A DESCRIPTION OR EXAMPLE OF THE SITUATION.

2. WHAT I HAVE TO DO.

   - WHAT CHANGES ARE REQUIRED?

3. HOW TO DO IT.

   - STEPS TAKEN.

4. WHAT DOES IT BUY ME.

   - THE THINGS I DIDN'T HAVE TO DO AND THE TIME SAVED.

III. CASE STUDIES

CASE 1. (WHILE LOOPS)

1. WHAT I'VE GOT.

FOLLOWING ARE 4 ELEMENTS OF A DATABASE. UNLESS STATED DIFFERENTLY THESE ELEMENTS WILL BE THE "WHAT I'VE GOT" IN ALL THE FOLLOWING CASES. TRY IMAGINING THAT THESE ARE REALLY 400 ELEMENTS INSTEAD OF 4...THIS IS KNOWN AS A WILLING SUSPENSION OF DISBELIEF - WHICH SHOULD BE DUCK SOUP TO THOSE OF YOU HAVING ANY CONTACT WITH THIRD PARTY SALES-PEOPLE.

```
/L ALL
    1       ;**************
    2       ELEMENT PROPER-TIE     9(008)V9(002(
    3       LABEL "PROPERTY VALUE"
    4       DESCRIPTION "VALUE OF PROPERTY"
    5
    6       **************
    7       ;**************
    8       ELEMENT PROPER-TIE     9(008)V9(002(
    9       LABEL "PROPERTY VALUE"
   10       DESCRIPTION "VALUE OF PROPERTY"
   11
   12       **************
   13       ;**************
   14       ELEMENT PROPER-TIE     9(008)V9(002(
   15       LABEL "PROPERTY VALUE"
   16       DESCRIPTION "VALUE OF PROPERTY"
   17
   18       **************
   19       ;**************
   20       ELEMENT PROPER-TIE     9(008)V9(002(
   21       LABEL "PROPERTY VALUE"
   22       DESCRIPTION "VALUE OF PROPERTY"
   23
   24       **************
```

2. WHAT I HAVE TO DO.

THERE ARE SEVERAL ERRORS HERE WHICH NEED CORRECTING, AND ELEMENTS 2 THRU 4 SEEM TO BE IDENTICAL TO ELEMENT 1 . . . REDUNDANCY IS FINE, BUT THIS SEEMS TO BE TAKING UNDUE RISC.

3. HOW TO DO IT.

WE ARE GOING TO TAKE ADVANTAGE OF THE ITERATIVE ABILITY OF THE WHILE LOOP TO PERFORM THE SAME TASK OR TASKS EVERY TIME THAT IT FINDS A STRING THAT WE WILL SPECIFY TO IT.

IT IS ALWAYS NICE TO "SET SHORT" SO THAT UNDUE COMMENTS ISSUED BY EDITOR ON YOUR PROGRESS ARE ABORTED, ALSO, IT SAVES TIME.

IF YOU THINK THAT MORE THAN 50 INSTANCES OF YOUR "FINDQ" EXIST INTHE EDITOR FILE FROM THE START OF THE WHILE LOOP, THEN USE THE "SET TIMES=N" COMMAND.

WHEREVER YOU HAPPEN TO BE SITTING IN THE EDITOR FILE IS WHERE THE WHILE LOOP WILL TAKE OFF (JUST LIKE A STAND-ALONE FIND COMMAND), SO YOU WILL USUALLY WANT TO "LIST 1" OR "LIST FIRST" TO TAKE CARE OF THE .1 LINE NUMBERS.

```
/SET SHORT
/SET TIMES=300
/11
    1      ;**************
/WHILE
/   FINDQ "ELEMENT'
/   M*/*+2
MODIFY     2
ELEMENT PROPER-TIE      9(008)V9(002(
                RTY
ELEMENT PROPERTY        9(008)V9(002(
                                    R)
ELEMENT PROPERTY        9(008)V9(002)

MODIFY     3
LABEL "PROPERTY VALUE"

MODIFY     4
DESCRIPTION "VALUE OF PROPERTY  "
                                  DD
DESCRIPTION "VALUE OF PROPERTY"

MODIFY     8
ELEMENT PROPER-TIE      9(008)V9(002(
                RPHONE
ELEMENT PROPER-PHONE    9(008)V9(002(
                        RX(010)
ELEMENT PROPER-PHONE    X(010)

MODIFY     9
LABEL "PROPERTY VALUE"
                DDDDDIPHONE
LABEL "PROPERTY PHONE"

MODIFY    10
DESCRIPTION "VALUE OF PROPERTY  "
             DDDDDDDDIPHONE:
DESCRIPTION "PHONE: PROPERTY  "
```

```
   MODIFY    14
   ELEMENT PROPER-TIE     9(008)V9(002(
                  RDATE
   ELEMENT PROPER-DATE    9(008)V9(002(
                          RDATE
   ELEMENT PROPER-DATE    DATE

   MODIFY    15
   LABEL "PROPERTY VALUE"
                 DDDDDDI: DATE ON MARKET
   LABEL "PROPERTY: DATE ON MARKET"

   MODIFY    16
   DESCRIPTION "VALUE OF PROPERTY  "
                DDDDDDIMARKET DATE
   DESCRIPTION "MARKET DATE OF PROPERTY  "

   MODIFY    20
   ELEMENT PROPER-TIE     9(008)V9(002(
                  RCITY
   ELEMENT PROPER-CITY    9(008)V9(002(
                          RX(050)
   ELEMENT PROPER-CITY    X(050)

   MODIFY    21
   LABEL "PROPERTY VALUE"
                 DDDDDDI:CITY
   LABEL "PROPERTY;CITY"

   MODIFY    22
   DESCRIPTION "VALUE OF PROPERTY  "
                DDDDDDDDICITY:
   DESCRIPTION "CITY:  PROPERTY  "

   *21*STRING NOT FOUND BEFORE LIMIT
   AT DEPTH 2
   /K
   BAYEX
   PURGE OLD?Y


/L ALL
    1      ;**************
    2      ELEMENT PROPERTY       9(008)V9(002)
    3      LABEL"PROPERTY VALUE"
    4      DESCRIPTION "VALUE OF PROPERTY"
    5
```

```
 6      **************
 7      ;**************
 8      ELEMENT PROPER-PHONE   X(010)
 9      LABEL "PROPERTY PHONE"
10      DESCRIPTION "PHONE: PROPERTY  "
11
12      **************
13      ;**************
14      ELEMENT PROPER-DATE    DATE
15      LABEL "PROPERTY:DATE ON MARKET"
16      DESCRIPTION "MARKET DATE OF PROPERTY  "
17
18      **************
19      ;**************
20      ELEMENT PROPER-CITY    X(050)
21      LABEL "PROPERTY:CITY"
22      DESCRIPTION "CITY:  PROPERTY  "
23
24      **************
```

4. WHAT IT BOUGHT ME.

IN THIS PARTICULAR CASE, THE METHOD BOUGHT ME NOT VERY MUCH, UNLESS, OF COURSE, I HAPPENED TO WANT TO MODIFY ALL THE ELEMENTS CONTAINING THE WORD "PHONE" PLUS THE NEXT TWO FOLLOWING LINES IN THIS 400 ELEMENT, 2000 LINES DATABASE. IN THAT CASE THE METHOD DOES ALL THE FINDS AND SETS UP ALL THE MODIFIES, THEREBY SAVING MANY KEYSTROKES AND RE-RE-KEYSTROKES.

CASE 2. (WHILE LOOPS)

1. WHAT I'VE GOT.

THE DABABASE I JUST MODIFIED.

2. WHAT I HAVE TO DO.

A DEFAULT HEADING MUST BE PUT INTO EACH ELEMENT STRUCTURE.

3. HOW TO DO IT.

HERE, WE SEE THE USE OF THE HOLD FILE, A VERY HANDY FEATURE OF EDITOR. IF YOU USE IT, REMEMBER TO FIRST CLEAR IT OUT BY JUST SAYING "HOLD".

```
/A.1
     .1    HEADING "BAYRUG MAT EXAMPLE"
     .2    //
...
/HOLD .1
HEADING "BAYRUG MAT EXAMPLE"
/L1
    1      ;**************
/WHILE
/   FINDQ "ELEMENT"
/   BEGIN
/      ADD *,HOLD,NOW

    2.1   HEADING "BAYRUG MAT EXAMPLE"
...
    8.1   HEADING "BAYRUG MAT EXAMPLE"
...
   14.1   HEADING "BAYRUG MAT EXAMPLE"
...
   20.1   HEADING "BAYRUG MAT EXAMPLE"
...
*21*STRING NOT FOUND BEFORE LIMIT
AT DEPTH 2
/K
BAYEX
PURGE OLD?Y


/L ALL
    .1    HEADING "BAYRUG MAT EXAMPLE"
    1     ;**************
    2     ELEMENT PROPERTY
    2.1   HEADING "BAYRUG MAT EXAMPLE"
    3     LABEL "PROPERTY VALUE"
    4     DESCRIPTION "VALUE OF PROPERTY"
    5
    6     **************
    7     ;**************
    8     ELEMENT PROPER-PHONE   X(010)
    8.1   HEADING "BAYRUG MAT EXAMPLE"
    9     LABEL "PROPERTY PHONE"
   10     DESCRIPTION "PHONE: PROPERTY  "
   11
```

```
   12       **************
   13       ;**************
   14       ELEMENT PROPER-DATE    X(050)
   15       LABEL "PROPERTY: DATE ON MARKET"
   16       DESCRIPTION "MARKET DATE OF PROPERTY  "
   17
   18       **************
   19       ;**************
   20       ELEMENT PROPER-CITY    X(050)
  20.1      HEADING "BAYRUG MAT EXAMPLE"
   21       LABEL "PROPERTY:CITY"
   22       DESCRIPTION "CITY:  PROPERTY  "
   23
   24       **************
/D .1
     .1     HEADING "BAYRUG MAT EXAMPLE"
/K
BAYEX
PURGE OLD?Y
```

4. WHAT DOES IT BUY ME.

I SAVED HAVING TO FIND THE LINE POSITION OF "ELEMENT" EACH TIME, AND DID NOT HAVE TO "ADD" THE NEW LINE.

CASE 3. (WHILE LOOPS)

1. WHAT I'VE GOT.

SAME AS ABOVE.

2. WHAT I HAVE TO DO.

EVERYWHERE THAT A "DATE" ELEMENT IS DEFINED, I NEED TO PUT IN A "FORMAT MMDDYY" STATEMENT IMMEDIATELY BEFORE THE DESCRIPTION STATEMENT.

NOTE THE USE OF MULTIPLE LOOPS, AND THE POSSIBILITIES THEY OFFER. ALSO, YOU MIGHT WISH TO DELETE THE LINE THAT YOU ARE SITTING ON AND REPLACE IT WITH WHAT IS IN THE HOLD FILE...AND THEN MAYBE PRINT OUT WHAT YOU HAVE JUST CHANGED WITH A "LIST*/*+4" COMMAND.

3. HOW TO DO IT.

```
/L ALL
    1       ;**************
    2       ELEMENT PROPERTY       9(008)V9(002)
    2.1     HEADING "BAYRUG MAT EXAMPLE
    3       LABEL "PROPERTY VALUE"
    4       DESCRIPTION "VALUE OF PROPERTY"
    5
    6       **************
    7       ;**************
    8       ELEMENT PROPER-PHONE    ...
/A.1
     .1     FORMAT MMDDYY
     .2     //
...
/HOLD .1
CLEAR HOLD? Y
FORMAT MMDDYY
/L1
    1      ;**************
/WHILE
/   FINDQ "-DATE"
/   BEGIN
/      FINDQ "DESCRIPTION"
/      LISTQ*-2
/      ADD *,HOLD,NOW
/      LISTQ *+2
/      END


HEADING "BAYRUG MAT EXAMPLE"
   15.1   FORMAT MMDDYY
...

*21*STRING NOT FOUND BEFORE LIMIT
AT DEPTH 2

/L ALL
     .1     FORMAT MMDDYY
    1       ;**************
    2       ELEMENT PROPERTY       9(008)V9(002)
    2.1     HEADING "BAYRUG MAT EXAMPLE"
    3       LABEL "PROPERTY VALUE"
    4       DESCRIPTION "VALUE OF PROPERTY"
    5
```

```
     6       **************
     7       ;**************
     8       ELEMENT PROPER-PHONE   X(010)
     8.1     HEADING "BAYRUG MAT EXAMPLE"
     9       LABEL "PROPERTY PHONE"
    10       DESCRIPTION "PHONE: PROPERTY  "
    11
    12       **************
    13       ;**************
    14       ELEMENT PROPER-DATE    DATE
    15       LABEL "PROPERTY: DATE ON MARKET"
    15.1     FORMAT MMDDYY
    16       DESCRIPTION "MARKET DATE OF PROPERTY  "
    17
    18       **************
    19       ;**************
    20       ELEMENT PROPER-CITY    X(050)
    20.1     HEADING "BAYRUG MAT EXAMPLE"
    21       LABEL "PROPERTY;CITY"
    22       DESCRIPTION "CITY:  PROPERTY  "
    23
    24       **************
/D.1
     .1     FORMAT MMDDYY
 /K
BAYEX
PURGE OLD?Y
```

4. WHAT IT BOUGHT ME.

- SAVED TWO POSITIONINGS OF THE FILE FOR EACH "DATE".
- SAVE AN "ADD" OF THE NEW LINE FOR EACH "DATE"

CASE 4. (WHILE LOOPS)

1. WHAT I'VE GOT.

SAME AS ABOVE.

2. WHAT I HAVE TO DO.

NOW THAT WE HAVE SET UP THE ELEMENTS NEEDED FOR THE DATA-BASE, WE ARE GOING TO WANT TO GROUP THEM TOGETHER IN RECORD FORMATS SO THAT THEY CAN BE PART OF AN APPLICATION SYSTEM. WE NEED TO BUILD SOME RECORD FORMATS WHICH WILL UTILIZE THE ELEMENT NAMES AS ITEM NAMES FOR THE RECORDS. I AM ONLY INTERESTED IN THE LINES WITH THE WORD/STRING "ELEMENT" IN THEM, AND WISH TO CREATE A FILE WITH THEM ALL TOGETHER, ONE AFTER THE OTHER, ADJACENT, CONTIGUOUS, ETC.

3. HOW I DID IT.

```
     1     ;**************
/WHILE
/    FINDQ "ELEMENT"
/    BEGIN
/       HOLDQ *,APPEND
/       LISTQ *+1
/       END
HOLD FILE LENGTH IS 1 RECORD
HEDING "BAYRUG MAT EXAMPLE"
HOLD FILE LENGTH IS 2 RECORDS
HEADING "BAYRUG MAT EXAMPLE"
HOLD FILE LENGTH IS 3 RECORDS
HEADING "BAYRUG MAT EXAMPLE"
HOLD FILE LENGTH IS 4 RECORDS
HEADING "BAYRUG MAT EXAMPLE"
*21*STRING NOT FOUND BEFORE LIMIT
AT DEPTH 2
/L LAST
    24     **************
/ADDQ 24,HOLDQ,NOW
...
LAST LINE =    28
```

```
/L ALL
    1      ;**************
    2      ELEMENT PROPERTY        9(008)V9(002)
  2.1      HEADING "BAYRUG MAT EXAMPLE"
    3      LABEL "PROPERTY VALUE"
    4      DESCRIPTION "VALUE OF PROPERTY"
    5
    6      **************
    7      ;**************
    8      ELEMENT PROPER-PHONE    X(010)
    9      LABEL "PROPERTY PHONE"
   10      DESCRIPTION "PHONE: PROPERTY  "
   11
   12      **************
   13      ;**************
   14      ELEMENT PROPER-DATE     DATE
  14.1     HEADING "BAYRUG MAT EXAMPLE"
   15      LABEL "PROPERTY: DATE ON MARKET"
  15.1     FORMAT MMDDYY
   16      DESCRIPTION "MARKET DATE OF PROPERTY  "
   17
   18      **************
   19      ;**************
   20      ELEMENT PROPER-CITY     X(050)
  20.1     HEADING "BAYRUG MAT EXAMPLE"
   21      LABEL "PROPERTY:CITY"
   22      DESCRIPTION "CITY:  PROPERTY  "
   23
   24      **************
   25      ELEMENT PROPERTY        9(008)V9(002)
   26      ELEMENT PROPER-PHONE    X(010)
   27      ELEMENT PROPER-DATE     DATE
   28      ELEMENT PROPER-CITY     X(050)
/KEEP RECORDS
/T RECORDS
/DQ 1/24
NUMBER OF LINES DELETED = 29
/L ALL
   25      ELEMENT PROPERTY        9(008)V9(002)
   26      ELEMENT PROPER-PHONE    X(010)
   27      ELEMENT PROPER-DATE     DATE
   28      ELEMENT PROPER-CITY     X(050)
/K
RECORDS
PURGE OLD?Y
PURGE OF OLD FILE NOT CONFIRMED - TEXT NOT KEPT
```

4. WHAT DOES IT BUY ME.

I DID NOT HAVE TO INDIVIDUALLY HOLD EACH LINE OF INTEREST IN A HOLD FILE OR DID NOT HAVE TO MANUALLY DELETE ALL UNWANTED LINES. I DID NOT HAVE TO KNOW OR CARE ABOUT OR LOCATE ANY OF THE LINES OF INTEREST.

CASE 5. (USE FILES)

1. WHAT I'VE GOT.

THERE IS A GROUP (POSSIBLY SURLY AND PRONE TO RIOT) CALLED BAYRUG WHICH HAS TWO FILES : BAYRUG87 AND BAYEX.

2. WHAT I HAD TO DO.

SOMEWHERE IN ONE OF THE FILES IS THE WORD BROGDIGNAGIAN ...FOR OBVIOUS REASONS THIS WORD MUST BE CHANGED TO "VERY LARGE". SINCE WE ARE IMAGINING THAT WE MIGHT BE DEALING WITH 400 FILES, THIS COULD TAKE SOME TIME IF WE HAD TO TEXT IN EACH FILE AND THEN FIND OUR WORD, THEN MODIFY THE LINE, THEN KEEP THE FILE.

3. HOW TO DO IT.

WITHOUT QEDIT OR MPEX OR SUPERMAN OR ROTO-ROOTER, THE ONLY SOLUTION IS IN THE CREATION OF USE FILES. IN THE FOLLOWING EXAMPLE, NOTE THAT THE USE FILES SHOULD ALWAYS BE KEPT UNNUMBERED. NOTE ALSO THAT THIS PROCEDURE COULD BE PUT INTO A JOB STREAM...MORE SPECIFICALLY:

```
JOB CARD
!EDITOR
USE USEFILEA
E
!EOJ
```

```
  :LISTF

  FILENAME

  BAYEX        BAYRUG87

  :FILE USEFILEA;REC=-80,1,F,ASCII;NOCCTL
  :LISTF;*USEFILEA
  :E
```

```
(C) HEWLETT-PACKARD CO. 1985
/T USEFILEA
FILE UNNUMBERED
/L ALL
    1
    2       FILENAME
    3
    4       BAYEX
    5       BAYRUG87
    6
    7
/DQ 1/3
NUMBER OF LINES DELETED = 3
/DQ 6/7
NUMBER OF LINES DELETED = 2
/C 1 TO "T " IN ALL
    4       T BAYEX
    5       T BAYRUG87
/C 20 TO ";USE USEFILEB ;K" IN ALL
    4       T BAYEX             ;USE USEFILEB ;K
    5       T BAYRUG87          ;USE USEFILEB ;K
/GQ ALL
/K
USEFILEA,UNN
USEFILEA ALREADY EXISTS - RESPOND YES TO PURGE OLD ANDKEEP NEW
PURGE OLD?Y
:E

HP32201A.07.17 EDIT/3000 SUN, APR 26,1987,  4:31 PM
(C) HEWLETT-PACKARD CO. 1985
/A
    1       SET SHORT
    2       SET TIMES=300
    3       WHILE
    4       FINDQ "BROBDIGNAGIAN"
    5       BEGIN
    6       M*
    7       L*-3/*+6
    8       END
    9       //
...
/KEEP USEFILEB,UNN
```

```
*21*STIRING NOT FOUND BEFOR LIMIT
AT DEPTH 4
BAYEX
PURGE OLD?
PURGE OF OLD FILE NOT CONFIRMED - TEXT NOT KEPT
MODIFY    33
LILLIPUTIAN TO BROBDIGNAGIAN) DATABASES ALONG WITTH THEIR
              DDDDDDDDDDDDDIVERY LARGE
LILLIPUTIAN TO VERY LARGE) DATABASES ALONG WITH THEIR

    31   MOST OF MY TIME IS SPENT IN THE CREATION AND MANIPULATION
    32   OF MEDIUM TO LARGE-SIZE (OR, FOR THE MORE TECHNICALLY MINDED:
    33   LILLIPUTIAN TO VERY LARGE) DATABASES ALONG WITH THEIR
    34   PROGRAMS.  ALTHOUGH THE INSTANCES ARE SO SELDOM AS TO ESCAPE
    35   MEMORY, IT SOMETIMES HAPPENS THAT THE WRONG THING HAS BEEN
    36   DONE, THAT THE RIGHT THING WAS NOT DONE, OR THE THING NEEDS
    37   TO BE DONE IN ANOTHER PLACE . . . AND RIGHT NOW.
    38
    39   WHAT IS NEEDED IN THESE CIRCUMSTANCES ARE WAYS TO CUT DOWN
    40   KEY-STROKES, SPEED SEARCHES, ACCOMPLISH GENERIC FILE MANIPU-
*21*STIRING NOT FOUND BEFORE LIMIT
AT DEPTH 4
BAYRUG87
PURGE OLD?
PURGE OF OLD FILE NOT CONFIRMED - TEXT NOT KEPT
/E
CLEAR? Y

END OF PROGRAM
:E

HP32201A.07.17 EDIT/3000 SUN, APR 26, 1987 4:33 PM
```

4. WHAT DOES IT BUY ME.

I GOT TO USE EDITOR IN A GENERIC FASHION TO DO WHATEVER FUNCTIONS ARE RESIDING IN USEFILEB. UNLESS I HAVE NOT YET READ THE EDIT-3000 MANUAL IN THE DIMENSION IT TRULY DESERVES, THIS GENERIC CAPABILITY IS NOT OTHERWISE AVAILABLE.

CASE 6. (HOLD FILES)

1. WHAT I'VE GOT.

THERE ARE FORTY FILES WHICH NEED TO BE COPIED TO ANOTHER ACCOUNT USING FCOPY.

2. WHAT I HAD TO DO.

COPY THE FILES.

3. HOW I DID IT.

USE THE HOLD COMMAND TO STORE AWAY THE PERTINANT PORTION OF USEFILEA CREATED IN CASE 5, THEN RESET THE MARGINS IN EDITOR TO DUMP THE HOLD FILE BACK INTO THE RIGHT-HAND PORTIION OF THE LINE. WILL WIND UP WITH THE "FROM" AND "TO" SECTIONS OF THE FCOPY COMMAND - SUITABLE FOR STREAMING. IT IS USEFUL TO REMEMBER THAT YOU CAN "HOLD" AND "HOLD,APPEND" WHATEVER LINES OR PORTIONS OF LINES (WITH THE SETS OF THE LEFT AND RIGHT MARGINS) DESIRED, AND THAT THE HOLD FILE WILL SURVIVE THE TEXTING IN OF ANOTHER FILE.

```
/T USEFILEA
FILE UNNUMBERED
/L ALL
    1     T BAYEX                  ;USE USEFILEB  ;K
    2     T BAYRUG87               ;USE USEFILEB  ;K
/CQ 20/60 TO "" IN ALL
/SET RIGHT=3
/CQ "T " TO "FROM=" IN ALL
/SET RIGHT=72
/L ALL
    1     FROM=BAYEX
    2     FROM=BAYRUG87
/HOLD 1/2
CLEAR HOLD?  Y
FROM=BAYEX
FROM=BAYRUG87
```

```
/L1
    1     FROM=BAYEX
/SET LEFT=20
/REPLACE 1/2,HOLD,NOW
    1
    1     FROM=BAYEX
    2
    2     FROM=BAYRUG87
/CQ "FROM" TO ";FROM" IN ALL
/CQ 50 TO ";NEW" IN ALL
/SET LEFT=1
/L ALL
    1     FROM=BAYEX              ;FROM=BAYEX              ;NEW
    2     FROM=BAYRUG87           ;FROM=BAYRUG87           ;NEW
/SET LEFT=20
/CQ "FROM" TO "TO" IN ALL
/SET LEFT=1
/L ALL
    1     FROM=BAYEX              ;TO=BAYEX                ;NEW
    2     FROM=BAYRUG87           ;TO=BAYRUG87             ;NEW
/CQ "           ;NEW" TO ".NUGROUP;NEW" IN ALL
    1     FROM=BAYEX              ;TO=BAYEX         .NUGROUP;NEW
    2     FROM=BAYRUG87           ;TO=BAYRUG87      .NUGROUP;NEW
/CQ "       ." TO "." IN ALL
/CQ "    ." TO "." IN ALL
/L ALL
    1     FROM=BAYEX              ;TO=BAYEX.NUGROUP;NEW
    2     FROM=BAYRUG87           ;TO=BAYRUG87.NUGROUP;NEW
/K
USEFILEA,UNN
USEFILEA ALREADY EXISTS - RESPOND YES TO PURGE OLD AND KEEP NEW
PURGE OLD?Y
```

4. WHAT IT BOUGHT ME.

- DID NOT HAVE TO TYPE IN 40 TO 400 MODS TO DESIGNATE THE "TO=" file-name WITH ALL THE WONDERFUL CHANCES TO MIS-READ AN ALPHA "O" FOR A NUMERIC 0 WHILE TYPING THE FILENAME.

CASE 7. (WORK-A-DAY CONSIDERATIONS)
or
(DAYS OF OUR LIVES)

THIS SECTION DEALS WITH SOME DAY-TO-DAY HINTS WHICH, IF YOU HADN'T ALREADY READ THIS FAR, YOU WOULDN'T GIVE A PFENNING FOR, BUT HERE THEY ARE.

1. IF ALL OF THE ABOVE GOOD WORKS FAILS, CONSIDER PUTTING THE LINES YOU WISH TO THROW INTO VARIOUS FILES INTO A FILE WITH A SHORT EASY-TO-TYPE NAME SUCH AS RR, WHICH YOU CAN JOIN (JOIN RR TO 100.1) QUICKLY TO ANY TEXTED IN FILE.

2. THE QUICK WAY TO RESEQUENCE YOUR FILES IS TO SAY "GQ ALL"

3. THE QUICK WAY TO LIST OR MODIFY LINES IS TO SAY L*/LAST OR M*/LAST.

4. EDITOR COMMANDS CAN BE STRUNG TOGETHER WITH SEMI-COLONS.
EX: C "AB" TO "BD" IN ALL;K

5. IF YOU ARE NOT INTO USING EXOTIC COMBINATIONS OF USEFILES AND WHILE LOOPS, YOU CAN AT LEAST TRY THE FOLLOWING:

```
WHILE
    FIND "BROBDINGAGIAN"
    M*
```

IF YOU DID A "LIST FIRST" BEFOREHAND, YOU WOULD BE LED THROUGH EACH OCCURRANCE OF THIS WORD IN THE FILE, AND HAVE A CHANCE TO CHANGE IT TO "VERY LARGE" OR NOT.

6. TO ADD NEW LINES IN THE MIDDLE OF A FILE WITHOUT THE BOTHERSOME GENERATION OF .1's OR POSSIBLLY .045's, CONSIDER A "GATHERQ" OF THE PORTION OF THE FILE FOLLOWING YOUR NEW ADDITIONS TO A HUNDRED OR SO LINES BEYOND...THIS WILL ALLOW ADDING LINES BY WHOLE NUMBERS AND HOLD OFF RESEQUENCING UNTIL YOU ARE THROUGH ADDING THE NEW LINES.
EXAMPLE:

```
1     LINE 1
2     LINE 2
3     LINE 3
/GATHERQ 2/3 TO 20
/ADD 2
```

IMPLEMENTING INFORMATION ACCESS

Sandy Lynch
Hewlett-Packard Company
Office Systems Division
8010 Foothills Blvd.
Roseville, CA 95678

INTRODUCTION

This paper will be a discussion on how Information Access has been implemented at Hewlett-Packard's Office Systems Division. More specifically, the focus will be on how productivity gains were made in Finance, Manufacturing, and Product Support by changing a process with Information Access. Finance has been able to use Information Access to gather information to be downloaded to a spreadsheet for further analysis in the process of inventory reconciliation. In Manufacturing, controllers use Information Access to track excess inventory, as well as to do ad hoc queries regarding particular part-numbers and vendors. Lastly, Product Support has been able to use Information Access and its Report Writer to automatically generate the required paperwork which triggers a distribution to customers on Software Subscription Services.

This paper will be a "how to" approach to optimally implementing Information Access. First, a product overview will be given followed by a discussion on how to plan an implementation. The steps to actually complete the implementation will be included as well. Lastly, each example will be discussed, including the actual steps that each department went through, samples of the actual tables and reports that were set up within Information Access, and any tradeoffs that were involved.

PRODUCT OVERVIEW

Information Access is an information retrieval system, consisting of HP Access on the PC and HP Access Central on the HP 3000, which gives PC users transparent access to IMAGE databases, as well as certain PC databases. PC users can combine and reformat the data they access and either download the results to their PCs or save them on the host HP 3000 for later use. Figure 1 provides a pictorial view of the product.

Information Access supports two types of data communication between the PC and the host - Basic Serial Networking and OfficeShare Networking.

Basic Serial Networking provides for the following PC to HP 3000 connections:

- Hardwired to an ATP on the HP 3000.
- Modem connection to the HP 3000.
- Statistical Multiplexer HP2334A or HP2334A Plus configuration to an ATP on the HP 3000 including certified (as listed on the HP2334A+ data sheet) X.25 public network connections.
- Cluster Controller HP2334A or HP2334A Plus configuration to an INP on the HP 3000 (does not include X.25 public network connections).

OfficeShare Networking provides for the following PC to HP 3000 connections:

- ThinLAN
- StarLAN
- SERIAL Network

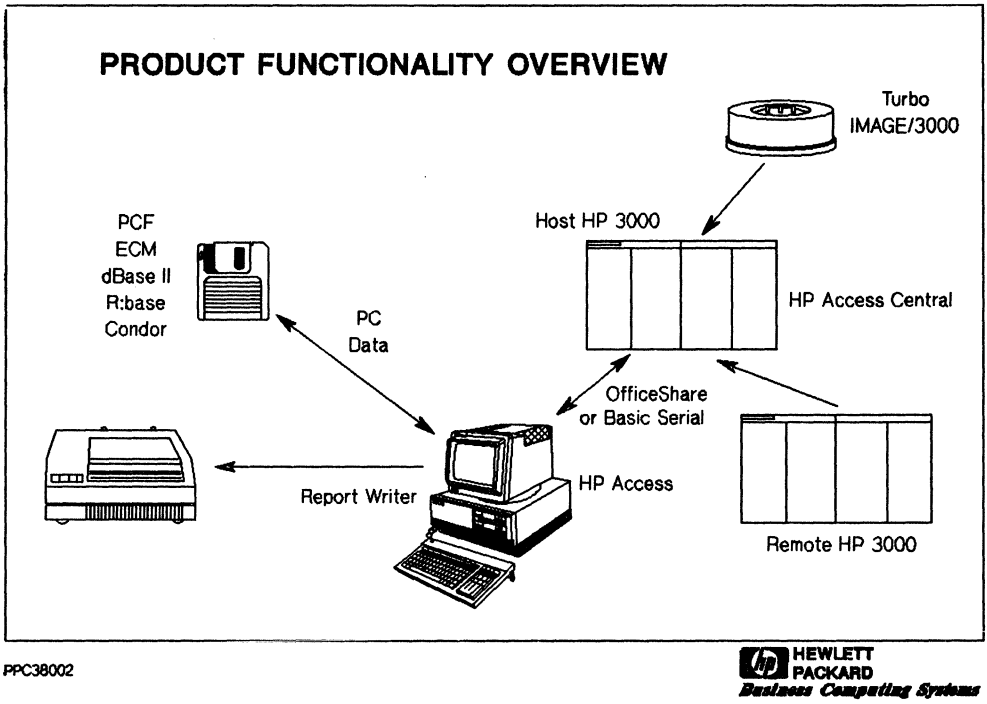


Figure 1. Information Access Overview

Figure 2 shows how the HP Access software interfaces with the HP Access Central software. The Database Administrator (DBA) controls which IMAGE data is available through the Administrator Utility of HP Access Central. The DBA provides further security by specifying which portion of that data the user can see. This information is stored in the HP Access Data Dictionary HDPDIC. HP Access users are not allowed to alter IMAGE databases in any way.

Data can be derived from host IMAGE databases and, if you've purchased the Remote IMAGE/3000 Link capability, from IMAGE databases located on remote HP 3000s one node removed from the host HP 3000. Note that host HP 3000 IMAGE databases will always be TurboIMAGE, assumming that the HP 3000 is running U-MIT or later. However, the Remote IMAGE/3000 Link can access IMAGE databases on systems one node away in either Turbo or non-Turbo format (any remote HP 3000 running pre-U MIT MPE would have non-TurboIMAGE databases.

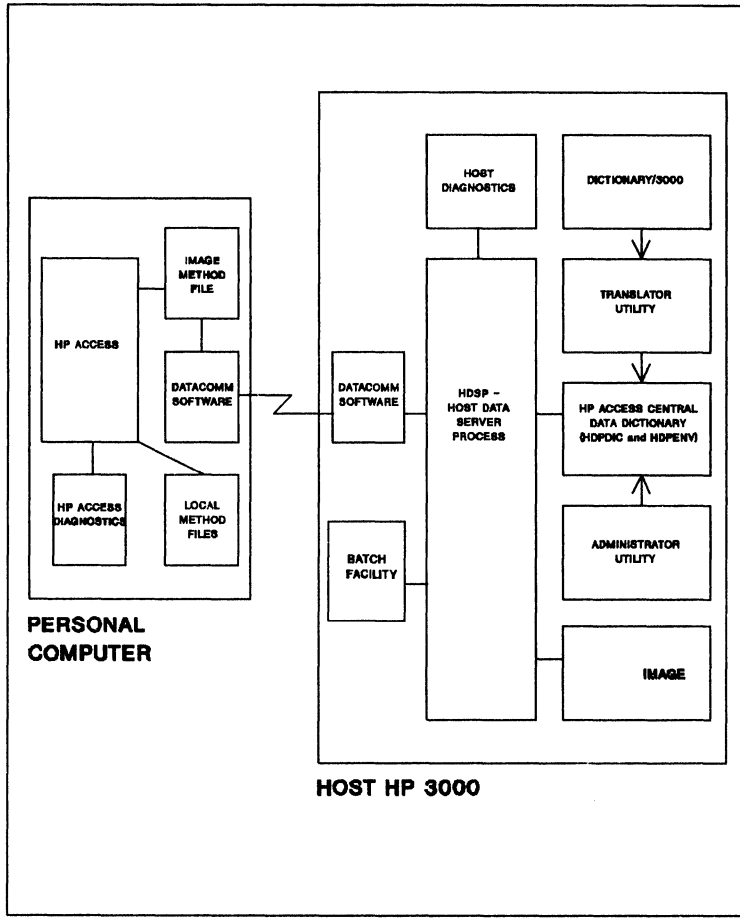


Figure 2. HP Access/HP Access Central Software Interface

PC users make use of HP Access Central when they run HP Access and select the Remote Tables function on the Main Menu screen. Together, HP Access and HP Access Central let the PC user:

- Access local PC databases (Condor, dBASEII, PCF, ECM, and R:base 4000/5000), host IMAGE databases, and remote IMAGE databases without needing to know how the databases are structured.
- Combine information from various IMAGE databases, independent of where the databases are located in the network. The HP 3000s can be connected using DS/3000, DS/X.25, or LAN.
- Access IMAGE data as relational tables (tables with rows and columns of data), a format familiar to most PC users.
- Use a menu-driven format to perform all operations.
- Select just the information needed for viewing and further manipulation.
- Sort and summarize table information.
- Save the results of local PC or HP 3000 table manipulations. Either type of manipulation may be saved locally on the PC or on the HP 3000.
- Reformat result tables for use in poplar PC applications. Tables can be converted to:
    - Condor, dBASEII, R:base, PCF, or ECM database formats.
    - DIF format for use in VisiCalc.
    - WKS format for use in Lotus 1-2-3.
    - A formatted ASCII file for use in WordStar, Executive MemoMaker, or some other PC editor.
    - QB (Quoted Basic) format for use in many custom applications.
- Reformat result tables for use in HP 3000 applications. Tables can be converted to:
    - DIF format for use in Deluxe VisiCalc/3000.
    - SD (self-describing) format for use in HP ListKeeper, DSG/3000, or HP EasyChart.
    - A formatted ASCII file for use in HP Word, TDP/3000, or some other HP editor.
- Use command files (PC or HP 3000 resident) to automate complex command sequences or to allow lengthy queries to be performed during off hours. HP 3000 resident command files provide a PC initiated batch capability, as the PC is free for other functions after the remote command file processing has been initiated.
- Generate sophisticated reports using Report Writer.

On the host HP 3000, HP Access Central provides:

- The HP 3000 software interface needed by HP Access to retrieve IMAGE data from the host HP 3000 and from remote HP 3000s.
- A Host Batch Facility, which allows execution of HP Access commands without the intervention of PC resident software. The Host Batch Facility can be run interactively, as a job stream on the HP 3000, or (as mentioned above) can be initiated from the PC.
- A data dictionary (distinct from Dictionary/3000 and System Dictionary/3000) in which the DBA defines (1) the IMAGE data available to PC users, (2) the users who can access the data using HP Access Central, and (3) what subset of the data each group of users is allowed to access.
- Two utilities, the Administrator Utility and the Translator Utility, which give the DBA a quick, friendly screen interface for performing all operations involving the data dictionary. These two utilities are the tools by which the DBA defines IMAGE data access for HP Access Central.
- A synchronization reporting function, which makes it easy for the DBA to detect discrepancies that may arise over time between the dictionary and IMAGE database structures.
- Diagnostics software to test the integrity of communications between HP Access on the PC and HP Access Central on the host HP 3000.

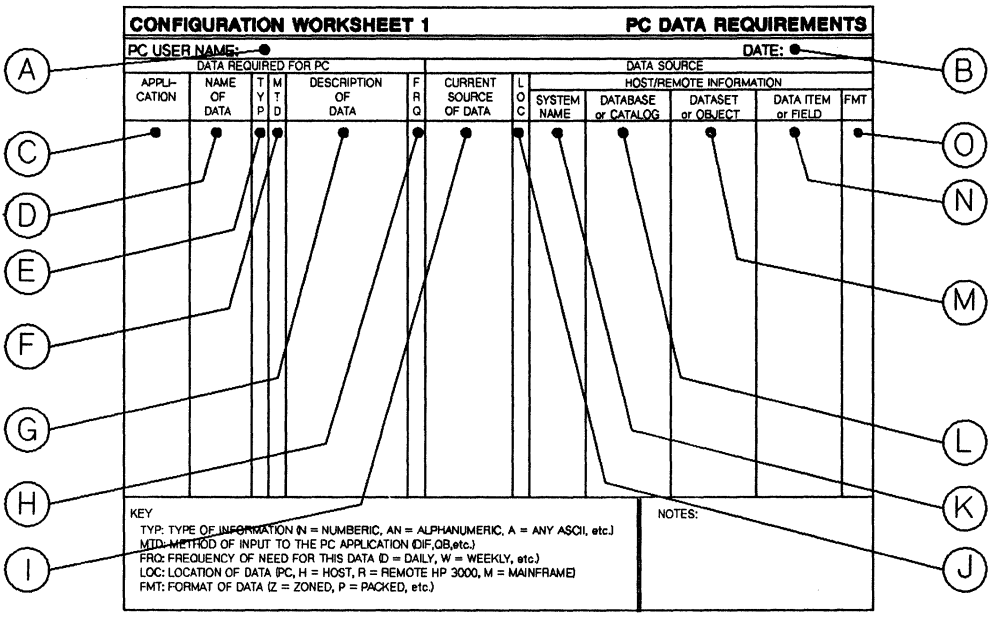
PLANNING YOUR CONFIGURATION

The planning process for implementing Information Access involves several steps, which are made easier by the use of designated worksheets to complete at each step. These planning stages are:

- Identifying your PC users and their data needs by using Worksheet 1 (PC Data Requirements).
- Using Worksheet 2 (Access Group Definition) to gather your PC users into access groups, arrive at user definitions, and list the configured tables that the group needs to access.
- Using Worksheet 3 (View Table Definition) to help you determine how your view tables should be configured.

The worksheets discussed above are included in the *HP Access Central: Planning and Configuring* manual, which is a part of the manual set for Information Access.

Worksheet 1: PC Data Requirements



Worksheet 1: PC Data Requirements

Worksheet 1 is used to gather detailed information from PC users about their current data needs. Have each PC user (or potential PC user) who accesses data of any kind, from any source, complete the worksheet. If the PC user is accessing HP 3000 data, the DBA may have to complete the last six columns.

WORKSHEET DESCRIPTION

At the top of the worksheet are spaces for the **PC User Name** (A) and the **Date** (B) the information was gathered.

The first six columns are labeled **Data Required for PC.** They indicate what data the PC user uses, some of the characteristics of the data, and how frequently the PC user needs it:

- **Application** (C) indicates what PC application (if any) the data is used in.
- **Name of Data** (D) is the name of the data item. (Take note of any limitations on item names in applications your PC users routinely use.)

- **TYP** (E) indicates the type of information. The worksheet suggests N for numeric, AN for alphanumeric, and A for a more general ASCII format. Don't feel confined to this suggested code.

- **MTD** (F) means the method by which the data is introduced into the PC application. Is it a DIF file for use in Visicalc? Is it a QB (Quoted Basic) file used in Lotus 1-2-3 or in R:base 5000? Or is it an ASCII file used in a PC editor?

- **Description of Data** (G) gives you some room to expand on the previous columns or to note anything special about the data item.

- **FRQ** (H) is the frequency of need for this data. On the average, how often does the PC user need this data? Every day, each week, once a month? The frequency of the data need will help you determine which PC users stand to benefit most from HP Access Central.

The last seven columns are labeled **Data Source**. They indicate where the PC user currently gets the required data and (for IMAGE data) what system, database, and dataset the data item resides in:

- **Current Source of Data** (I) indicates where the user is currently getting the data from. The data source may be a utility the user runs, a custom program, or the user may be handed this information as a report generated by a programmer.

- **LOC** (J) is the location of the data. The worksheet suggests L for local data on the PC itself, H for data from the host HP 3000, and R for data from remote HP 3000s. Feel free to modify this suggested code to something more directly applicable to your computing environment.

- **System Name** (K) means the name of the computer the data item resides on. For remote HP 3000s, be sure to include the device class name (for a DS/3000 connection between the host and remote machines) or the node name, qualified, if necessary, by the domain and organization (for an NS/3000 connection), under which the system is configured for MPE. You'll be using this information to define the remote system in the HP Access Central data dictionary.

- **Database Name** (L) means the name of the IMAGE database containing the data item. You may also want to include the MPE group and account for the database, and perhaps the user class through which this PC user is allowed to view the database.

- **Dataset Name** (M) means the IMAGE dataset which contains the data item.

- **Data Item Name** (N) is the name of the data item in the dataset.

- **FMT** (O) represents the format of the data item. You use this column to catch items of type zoned decimal or packed decimal, which will need to be converted for use in HP Access Central.

RESULTS

Once Worksheet 1 has been collected from your PC users, you'll have a good idea which PC users should be configured as Information Access users. It will probably also be clear, because of their common data needs, which users belong together in one access group. Because you've determined the source of the data your PC users need, you'll also have a good idea what IMAGE data needs to be defined in HP Access Central. You will know which remote systems, which databases, and which datasets need to be configured as IMAGE tables for use in HP Access Central.

If your users plan to use Information Access output in their PC applications, you will need to be aware of any special formats or naming conventions the data must conform to. Here are some things to be on the lookout for:

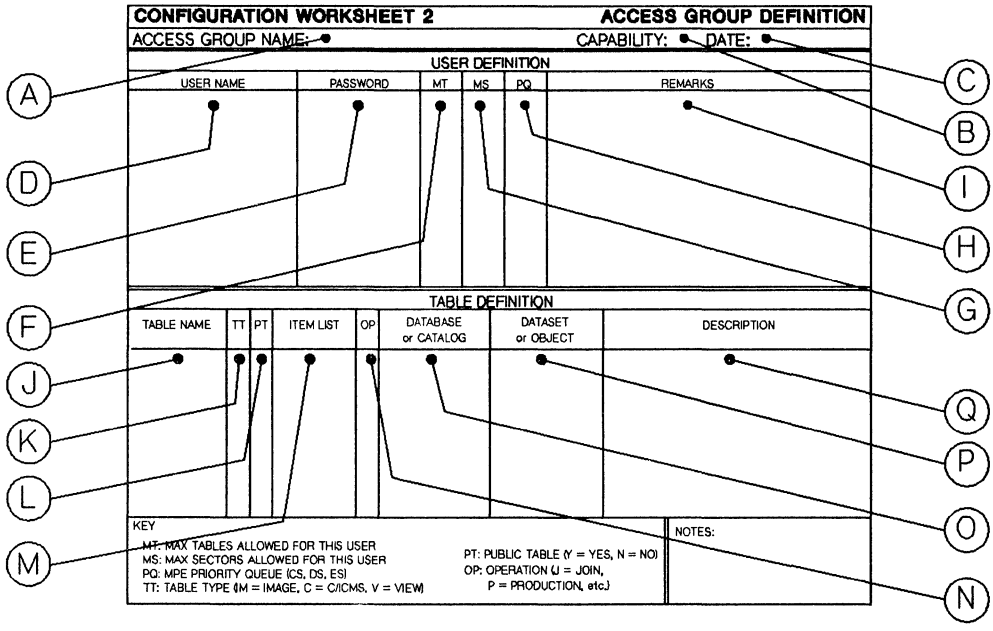
- Is the maximum length of item names and table names less than the 16 characters allowed by HP Access Central? Are the naming conventions more restrictive? If so, you need to configure your tables accordingly.

- What data types does the PC application support? You don't want to include in your table an item whose data type is unsupported.

- If the PC application accepts real numbers, does it require the decimal point always to be in the same place? If so, this requirement will affect how you configure your tables.

Your PC users will probably be able to alert you to this sort of thing for the PC applications they use. If not, consult the reference manuals for the products in question.

Worksheet 2: Access Group Definition



Worksheet 2: Access Group Definition

Worksheet 2 is used to arrive at the definitions of access groups, the definitions of the users who will belong to them, and the definitions of the IMAGE and view tables each access group will be able to access. On each worksheet, fill in an appropriate name and capability for the access group. For any users you want in the access group, create a user name and password.

WORKSHEET DESCRIPTION

At the top of the worksheet are spaces for the **Access Group Name** (A), the **Capability** (B) of the access group, and the **Date** (C) you defined the access group.

Five columns are provided for the definitions of Information Access Users belonging to the access group:

- **User Name** (D) is the name you want to give the HP Access Central user. The name can be up to 16 characters long and must begin with an alpha character followed by any combination of alphanumeric characters and/or hyphens.
- **Password** (E) is the password (optional) you want to give the user. If you provide a password, it too can be up to 16 characters long and must begin with an alpha character followed by any combination of alphanumeric characters and/or hyphens.
- **MT** (F) is the maximum number of result tables this user can save on the host HP 3000. You might put UNLMT in this column if you don't want to set a limit.
- **MS** (G) is the maximum number of disc sectors this user's saved tables can occupy on the disc drives connected to the host HP 3000. You might put UNLMT in this column if you don't want to set a limit.
- **PQ** (H) is the MPE priority of the user's server process. The default is CS, the normal priority for interactive processes. DS is a lower priority and ES is lower still.
- **Remarks** is a column for any additional information about this user. (Unlike the other columns, these remarks are not part of the user definition and are not recorded in the data dictionary.) The PC user's full name might go here, for example. Or, since there need not be a one-to-one correspondence between user names and real people, you might indicate which people are using which user name.

Eight columns are provided for the definitions of configured tables accessible to users in this access group:

- **Table Name** (J) lets you assign a name to the IMAGE or view table being described. Table names can be up to 16 characters long and must begin with an alpha character followed by any combination of alphanumeric characters and any of these characters: + - * / ? ' # % & @. (In naming tables, keep in mind any limitations on table names in the applications your PC users plan to use with data saved from HP Access Central.)
- **TT** (K) indicates the table type, where IM stands for an IMAGE table (a table derived directly from one dataset) and V stands for a view table (A view table (a table derived from one or more previously configured tables).
- **PT** (L) indicates whether the table is a public table. Because no access groups are assigned to it, a public table can be accessed by any Information Access user. A "No" in this column means you will need to configure table security for this table. (NOTE: A table can be a public table and still have item security on selected items. Keep track of item security in the Description column.)
- **Item List** (M) is where you list the names of the items contained in the table. A configured table can have up to 64 items (columns). For IMAGE tables, the ADDIMAGETABLE screens

provide an easy way to include all items from a dataset, so you may not need to list all the items here in that case.

- **OP** (N) indicates, for view tables, the kind of operation required to configure the table. J3, for example, might mean three tables are JOINed to create the view table. For a view table derived from one IMAGE table, you might use an N to mean no special operation is required.
- **Database Name** (O) is the name of the IMAGE database the item on the same line comes from.
- **Dataset Name** (P) is the name of the dataset the item comes from.
- **Description** (Q) lets you describe anything noteworthy about the item, or refer to a more detailed definition of the view table in Worksheet 3.

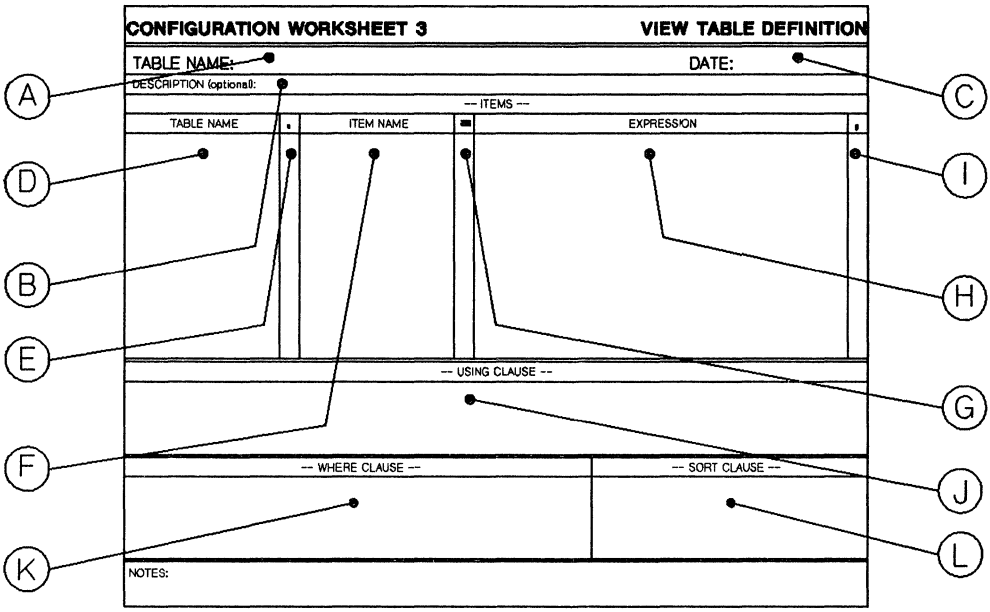
RESULTS

Once you have completed Worksheet 2 for each access group along with Worksheet 3 (described below) for each view table, you'll have all the information required to configure access groups, users, IMAGE and view tables, and table and item security.

Worksheet 3: View Table Definition

Worksheet 3 is used to work out complete definitions of all view tables you want to configure in the Administrator Utility of HP Access Central. Complete this worksheet for each view table.

WORKSHEET DESCRIPTION



Worksheet 3: View Table Definition

At the top of the worksheet are spaces for the **Table Name** (A), the **Description** (B), and the **Date** (C) you created the view table definition.

Six columns are provided for the definition of items, though in most cases you will only need one column, the Item Name:

- **Table Name** (D) is the name of the table from which the item comes. The table name is ordinarily not required. If two tables being JOINed both have the item ITEM1 in them, then TABLE1.ITEM1 designates which table to draw ITEM1 from and retains that item name as the column heading in the view table.
- The period (E) is needed only when the Table Name column is used.
- **Item Name** (F) is the name the item will have in the view table. There are three ways to represent an item:
    - If the item name occurs only once in the table being combined and you don't want

to rename the item in the view table, fill in the Item Name column only.

- If the item name occurs more than once in the tables being combined and you don't want to rename the item in the view table, fill in the first three columns (Table Name through Item Name) only.

- If you want to rename the item and/or arithmetically manipulate it, fill in the last three columns (Item Name through Expression) only.

- The equal sign (G) is needed only when the Expression column is used.

- **Expression** (H) can be used to fully qualify an item if its Item Name is not unique (and the Table Name column as not been used to qualify it). It can also be used to perform some arithmetic operations on the data item. You can, for example, convert an annual amount (in the source table) to a monthly amount (in the view table) by indicating division by 12.

- The comma (I) is used to separate items in the Items clause.

The **Using Clause** (J) is used to identify the configured table(s) from which the view table is derived. View tables can be derived from a single table or view tables can be derived from two, three, or four tables using the PRODUCTION and/or JOIN functions. JOINs can be done on one to four items from each table.

The **Where Clause** (K) is used to specify the type or range of records you want included in the view table. The operators you can use in this clause are =,<,>,<=,>=, and <>. You can also combine these operators using the AND and OR functions. The MATCH operator can be used in conjunction with wildcard characters to narrow the range of records still further.

The **Sort Clause** (L) is used to specify up to eight items you want the records sorted on, once the Item, Using, and Where clauses have determined which records are in the view table. The default sort is Ascending, but you can specify the type of sort with an A (Ascending) or a D (Descending)

CONFIGURING INFORMATION ACCESS

Configuring Information Access is done in several phases which involve two sorts of activity. The first is using your Configuration Worksheets and the Administrator Utility to add appropriate entries to the data dictionary. Following are the steps you would go through in the Administrator Utility to complete this:

1. **Configure Remote Systems.** Remote Systems are configured by providing a name for the remote system, a node name (the name of the system according to DS or NS convention), and logon information for each remote system you want to configure.

2. **Configure Host and Remote Databases.** To configure any required databases, provide a database name, the MPE group and account it resides in, the remote system name (if it is not on the host), the database password, and the open mode (if different from the default).

3. **Configure IMAGE Tables.** When you have successfully configured all required IMAGE databases, you are now ready to configure IMAGE tables from them. An IMAGE table is derived from one IMAGE dataset and can contain as may as 64 items (columns). After specifying which database you want to derive IMAGE tables from, you will be prompted with all datasets in that particular database, one by one. You may either select the items you want

from a dataset, or you may simply proceed to the next dataset, if you do not want a particular one.

4. **Configure View Tables.** View tables are configured after all IMAGE tables have been configured. View tables are derived from one to four previously configured tables, and can contain as many as 64 items. View tables are defined on a screen much like Worksheet 3. You provide a table name, the items to be included in this view table, which tables this view table is derived from and how the tables are to be joined together (JOIN or PRODUCTION), if any sub-setting of particular values is to be be done, and if any sorts are to be applied.

5. **Configure Access Groups.** At this point, you are ready to configure the access groups through which you control data security and into which you will be adding users.

6. **Configure Users.** Once access groups are configured, you are ready to configure users to put in them. The information you provide for each user is a user name, user password, the access group you want to associate with this user, the limits (if any) on the number of tables the user can save, the number of disc sectors those saved tables can occupy, and the server process's MPE priority (if different from the default of CS).

7. **Configure Table Security.** Table security is configured by assigning access groups to IMAGE and view tables. There is virtually no limit to the number of access groups you can assign to a table. Any tables which have no access groups assigned to them, will remain as **public tables.**

8. **Configure Item Security.** Security can be assigned to particular items by assigning access groups to items in the configured tables. You can assign as many as eight access groups to an item. Any items which have no access groups assigned to them remain as **public items,** and are accessible by any user who is allowed into the table which contains the item.

The second step involved in configuring Information Access is verifying that the configuration is correct. This can be done in two ways:

- **Generate a report.** The Report feature of the Administrator Utility can be used to print a complete report on how Information Access has been configured, including all remote systems configured, all databases configured, all IMAGE and view tables configured, and all users associated with an access group.

- **Run HP Access.** Run HP Access and for one user in each access group you have configured, type in that user name and password and select Remote Tables. Check to see that the correct configured tables appear for that user. If you have added item security which does not allow this user to see a certain item, you might select the table in which the item occurs to verify that the item is indeed hidden from this user.

At this point you have completely configured Information Access and PC users will be able to access any configured IMAGE data.

EXAMPLE - INFORMATION ACCESS AND FINANCE

Introduction

Several workgroups at OSD worked on projects which investigated how they could use OSD Office Products (Information Access, Resource Sharing, and Print Central) to improve productivity. One of these workgroups, Finance, began their project by investigating their current work environment and identifying processes that were good candidates for automation. They focused on tasks that were manual, repetitive, and often involved gathering data from several different HP 3000 applications for further analysis.

In the investigation phase of their project, Finance characterized their work environment by having project team members keep records of all tasks involving data retrieval and information output for one month. The particular period selected included a month-end, which is a major activity for this group.

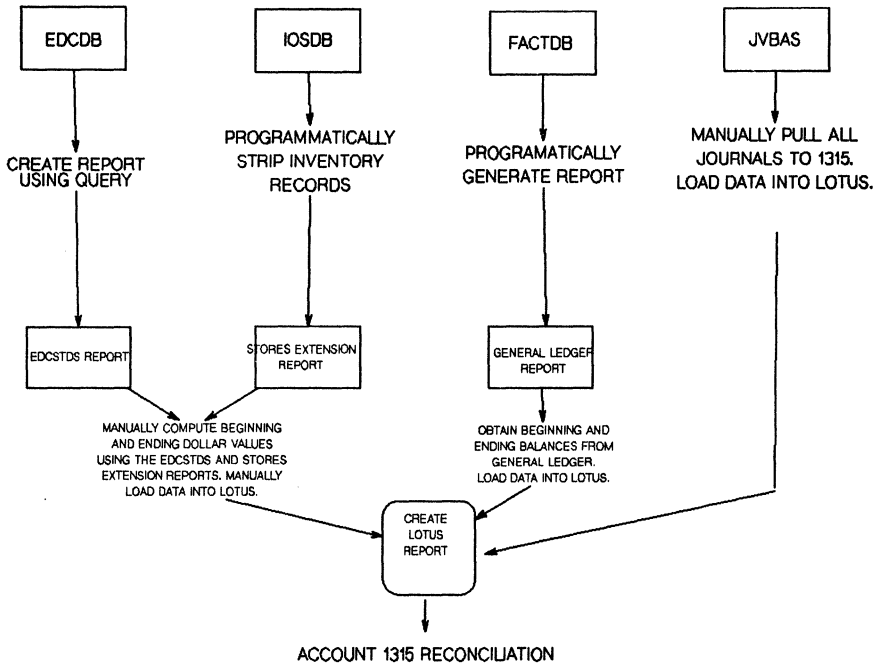
In characterizing their work environment, the project team discovered that Finance does a considerable amount of manual processing and re-keying of data between people and applications. Several processes were identified that offered good productivity paybacks for automation. The process to be discussed here is inventory reconciliation, which previously involved producing several reports and using the information from these reports to update Lotus 1-2-3 spreadsheets.

Original Process Flow

The figure shown below illustrates the process that was used for inventory reconciliation (for a particular account) prior to implementing Information Access.

PROCESS FLOW FOR INVENTORY RECONCILIATION

(PRIOR TO USING INFORMATION ACCESS)



As can be seen from the above illustration, information was retrieved from three databases (EDCDB, IOSDB, and FACTDB) to generate three separate reports. All of this information was entered manually into a Lotus 1-2-3 spreadsheet. All journals (invoices) for account 1315 were also manually loaded into the spreadsheet, and this provided the information to complete the account reconciliation for account 1315.

Planning the Configuration

The first step in planning the configuration was actually completed during the investigation phase of the project when each project member kept records of the tasks they did involving data retrieval and output. This data they kept is essentially the same information as would be filled in on Worksheet 1, so the next step was to determine what IMAGE tables and view tables were required to completely reconcile account 1315. As can be seen from the illustration shown above of the data flow, and mentioned above, there are three steps to the process, each of which requires specific tables. The three steps are described below, along with the IMAGE and view tables which are required at each step.

Step 1

Step 1 of the process is to join data from IOSDB and EDCDB, which is then output to a Lotus Transfer File. This file is then imported into Lotus 1-2-3 and used in formulas which compute the beginning and ending dollar values of the actual inventory in the 1315 account. The IMAGE datasets and data items required are:

- ITEM-DATA - from the EDCDB database. The data items required are PART-NUMBER and STD-OEM-COST-IC.
- STOCK-ACTIVITY - from the IOSDB database. The data items required are PART-NUMBER, SCR-NAME, LOCATION, ORDER-NUMBER, DATE-INV-ACT, QTY-ENTERED, S-C-S-2, AND INV-S-C-S-2.

The view table which actually joins the required data from the above IMAGE tables is defined on Worksheet 3 below. This view table takes advantage of a JOIN between two tables from different databases to define a new data item, EXT-STD. This table also uses the Where clause to subset the data so only a particular set of part-numbers are retrieved (those in account 1315).

CONFIGURATION WORKSHEET 3 — **VIEW TABLE DEFINITION**

TABLE NAME: MM1315 — DATE: 05/87

DESCRIPTION (optional): COMPUTES BEGINNING AND ENDING DOLLAR VALUES FOR 1315

-- ITEMS --

TABLE NAME	.	ITEM NAME	=	EXPRESSION	,
		PART#	=	STOCK-ACTIVITY.PART-NUMBER	
		SCR-NAME			
		LOCATION			
		ORDER#	=	ORDER-NUMBER	
		DATE	=	DATE-INV-ACT	
		QTY	=	QTY-ENTERED	
		EXT-STD	=	STD-OEM-COST-IC * QTY-ENTERED	
		FROMLOCN	=	S-C-S-2	
		TOLOCN	=	INV-S-C-S-2	

-- USING CLAUSE --

STOCK-ACTIVITY<PART-NUMBER> JOIN ITEM-DATA-HOST<PART-NUMBER>

-- WHERE CLAUSE --	-- SORT CLAUSE --
PART-NUMBER="30161-60001" OR PART-NUMBER="30456-60001" OR PART-NUMBER="30466-60001"	PART-NUMBER A

NOTES:

Step 2

The second step is to obtain information which is also output to a Lotus Transfer File, imported into Lotus 1-2-3, and used in formulas to calculate the beginning and ending balances from the General Ledger. An IMAGE table must be configured from the ALLFACT-DETAIL dataset of the FACTDB database. The data items to be included from the ALLFACT-DETAIL dataset are shown below in the worksheet. Note that in the actual view table DEBIT-FM1-US and CREDIT-FM1-US are for the first month, DEBIT-FM2-US and CREDIT-FM2-US are for the second month, and so on for all twelve months. The view table definition below takes advantage of the MATCH operator, which checks a character string for a particular value. In this case, the view table checks ACCOUNT for the value "1315@", where @ is a wildcard value, and keeps only these records.

CONFIGURATION WORKSHEET 3				**VIEW TABLE DEFINITION**	
TABLE NAME: GL1315				DATE: 05/87	
DESCRIPTION (optional): OBTAINS BEGINNING AND ENDING BALANCES FROM GEN LED					
		-- ITEMS --			
TABLE NAME	.	ITEM NAME	=	EXPRESSION	,
		ACCOUNT CHG-DEPT DEBIT-FM1-US CREDIT-FM1-US DEBIT-FM2-US CREDIT-FM2-US CONT.			
-- USING CLAUSE --					
ALLFACT-DETAIL					
-- WHERE CLAUSE --				-- SORT CLAUSE --	
ACCOUNT MATCH "1315@"				ACCOUNT A, CHG-DEPT A	
NOTES:					

Step 3

The third step of the process pulls all journals for the 1315 account out of a dataset JV-DETAIL, from the database JVBAS. In the original process flow, this database was not utilized as the journals were manually pulled to retrieve the information. The worksheet shown below indicates which data items are to be included from the dataset. This view table uses the Where clause to subset the data so that only information pertinent to account 1315 is retrieved. This table also uses the SUB function to extract several pieces of information from a single field, DETAIL-DESC.

CONFIGURATION WORKSHEET 3				VIEW TABLE DEFINITION
TABLE NAME: JV1315				DATE: 05/87
DESCRIPTION (optional): GETS ALL JOURNALS (INVOICES) FOR ACCOUNT 1315				
		-- ITEMS --		
TABLE NAME	.	ITEM NAME	=	EXPRESSION
		PART#	=	SUB(DETAIL-DESC,6,11)
		MO	=	SUB(DETAIL-DESC,4,2)
		JV-QTY	=	SUB(DETAIL-DESC,1,3)
		JV#/TR#	=	VOUCHER
		DEPT	=	CHG-DEPT
		AMOUNT		
		SRCR-FLAG		
-- USING CLAUSE --				
JV-DETAIL				
-- WHERE CLAUSE --			-- SORT CLAUSE --	
CHG-ACCT="1315"			PART# A, JV#/TR#	
NOTES:				

Configuration Of Information Access

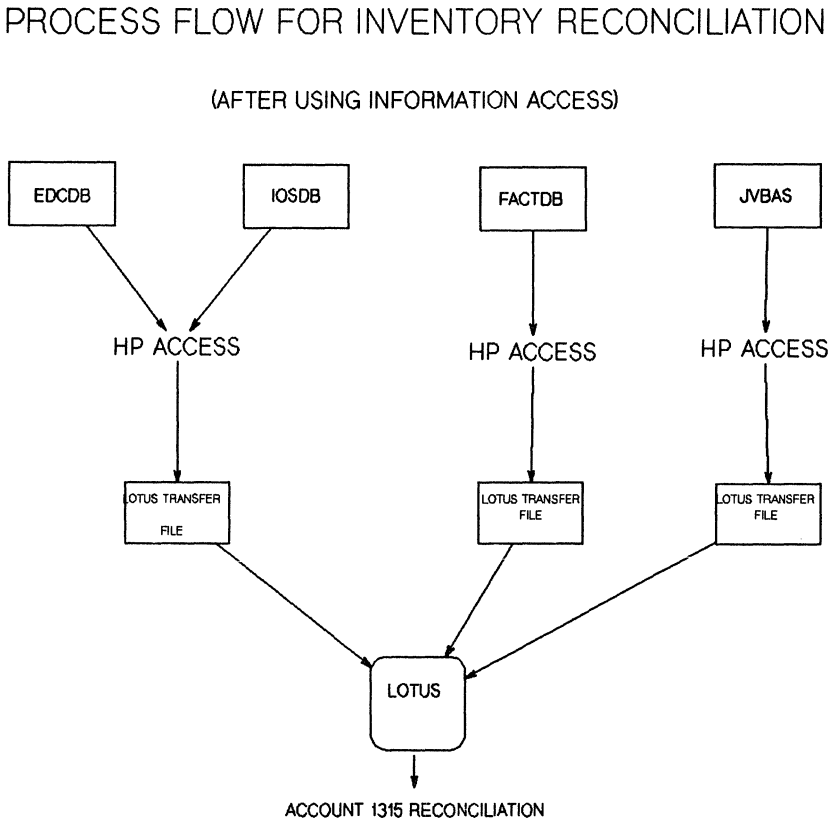
From the Administrator Utility, within Information Access, the following steps were completed:

- The databases EDCDB, IOSDB, FACTDB, and JVBAS were configured.
- The IMAGE datasets configured were ITEM-DATA from EDCDB, STOCK-ACTIVITY from IOSDB, ALLFACT-DETAIL from FACTDB, and JV-DETAIL from JVBAS.
- Using the three worksheets shown above for the view tables, three view tables were configured.

- An access group, FINANCE, was created, along with a user and password.
- The access group, FINANCE, was assigned to the IMAGE and view tables to provide table security.

Productivity Gains

Shown below is what the process flow for account 1315 reconciliation looks like after implementing Information Access.



Prior to using Information Access, the entire reconciliation process took approximately twelve hours to complete. With Information Access in place, this process takes about two hours, for a significant time savings.

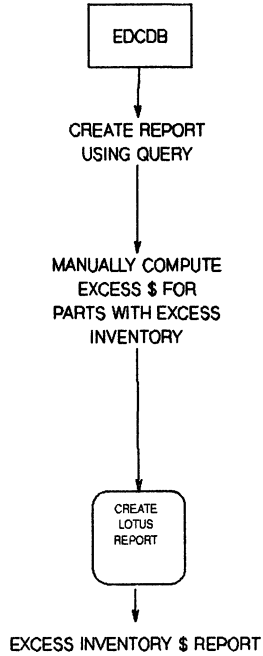
EXAMPLE - INFORMATION ACCESS AND MANUFACTURING

Introduction

With the advent of Just In Time Manufacturing, an area which had a lot of attention focused on it was the inventory dollar level of component parts. Of special concern were those components whose inventory level exceeded 52 weeks. The controllers for components would calculate excess inventory dollars (defined as greater than one year's supply) by generating a QUERY report of total inventory, grouped by controller, from the MM database, EDCDB. Each part-number that had greater than 52 weeks supply of inventory was highlighted, and the excess dollars were then calculated manually. This information was then entered into a spreadsheet and totaled for each controller. Because the information was being retrieved from an IMAGE database, and then manipulated manually because QUERY could not extract the desired information, the controllers investigated using Information Access as a way to improve the process of calculating excess inventory dollars. A diagram of the process flow for calculating the excess inventory dollar report, prior to using Information Access, is shown below.

PROCESS FLOW FOR EXCESS INVENTORY $ REPORT

(PRIOR TO USING INFORMATION ACCESS)



Planning The Configuration

The IMAGE datasets required were ITEM-DATA, from EDCDB, and INV-MASTER, from IOSDB. The data items required from ITEM DATA were PART-NUMBER, DESCRIPTION, CTLR, SIX-MO-RQMT, PART-CLASS, STD-MATL-COST, STD-TL-LBR-COST, STD-LL-LBR-COST, STD-TL-OVH-COST, STD-LL-OVH-COST, ACCT, AND A-B-C. The data items required from INV-MASTER were QTY-ON-HAND and QTY-IN-INSP.

Calculating the excess inventory dollars required two levels of view tables, both of which are shown below:

CONFIGURATION WORKSHEET 3 — **VIEW TABLE DEFINITION**

TABLE NAME: EXCESS1 — DATE: 05/87

DESCRIPTION (optional): PRELIMINARY TABLE FOR EXCESS-FINAL

-- ITEMS --

TABLE NAME	.	ITEM NAME	=	EXPRESSION	,
ITEM-DATA	.	PART-NUMBER			
		CTLR			
		SIX-MO-RQMT			
		ON-HAND-Q	=	(QTY-ON-HAND+QTY-IN-INSP)	
		WEEKS-OF-SUP	=	(QTY-ON-HAND+QTY-IN-INSP) / (SIX-MO-RQMT/26)	
		STD-COST	=	CASE PART-CLASS='P':STD-MATL-COST, ELSE (STD-MATL-COST+STD-TL-LBR-COST+ STD-LL-LBR-COST+STD-TL-OVH-COST)	

-- USING CLAUSE --

ITEM-DATA<PART-NUMBER> JOIN INV-MASTER<PART-NUMBER>

-- WHERE CLAUSE --	-- SORT CLAUSE --

NOTES:

CONFIGURATION WORKSHEET 3				VIEW TABLE DEFINITION
TABLE NAME: EXCESS-FINAL				DATE: 05/87
DESCRIPTION (optional): EXCESS $ BY CONTROLLER DATA				
-- ITEMS --				
TABLE NAME	.	ITEM NAME	=	EXPRESSION
		PART-NUMBER		
		CTLR		
		SIX-MO-RQMT		
		ON-HAND-QTY		
		ON-HAND-$	=	(ON-HAND-Q*STD-COST)
		EXCESS-$	=	((WEEKS-OF-SUPPLY-52)*(ON-HAND-Q* STD-COST))/WEEKS-OF-SUPPLY
		WEEKS-OF-SUP		
-- USING CLAUSE --				
EXCESS1				
-- WHERE CLAUSE --				-- SORT CLAUSE --
WEEKS-OF-SUP > 52				
NOTES:				

The first view table, EXCESS1, takes advantage of the CASE statement which allows new fields to be defined based on the information contained in another field. In this table, the CASE statement allows for differentiation of standard-cost for purchased and fabricated parts. The second view table, EXCESS-FINAL, uses the Where clause to allow selection of those records which meet the over one year's supply specification, thereby reducing considerably the number of records that must be retrieved.

Configuration Of Information Access

From the Administrator Utility, the following steps were completed:

- The IMAGE dataset, INV-MASTER, was configured as an IMAGE table. Note that the database, EDCDB, and the dataset, ITEM-DATA, were configured in the first example so they did not need to be configured again.
- Using the two worksheets shown above, two view tables were configured.
- The access group, CONTROL, was configured, along with a user name and password that each controller could use.
- The access group, CONTROL was assigned to the newly defined IMAGE and view tables to provide table security.

Using The Report Writer

A report format was designed to provide the controllers with appropriate information relating to the part-numbers which had greater than a 52 week supply. In the report format shown below, the highlighted fields indicate which fields will actually draw data from the HP Access table. The remaining text is simply a permanent part of the report format.

```
EXCESS DOLLARS REPORT
DATE
TIME

CTLR

PART-NUMBER SIX-MO-RQMT STD-COST ON-HAND-Q ON-HAND-$ WEEKS-OF-SUPPLY EXCESS-$
-------------------------------------------------------------------------------

PART-NUMBER SIX-MO-RQMT STD-COST ON-HAND-Q ON-HAND-$ WEEKS-OF-SUPPLY EXCESS-$


CTLR  EXCESS DOLLARS=SUM-OF-EXCESS-$
```

A couple points of interest in the above report format are the DATE and TIME fields, as well as the SUM-OF-EXCESS-$ field. The DATE and TIME fields are special fields which are defined within Report Writer and print the current date and time as the report is printed. The SUM-OF-EXCESS-$ is a statistical field which is actually a summation of the EXCESS-$ field for all PART-NUMBERs.

Productivity Gains

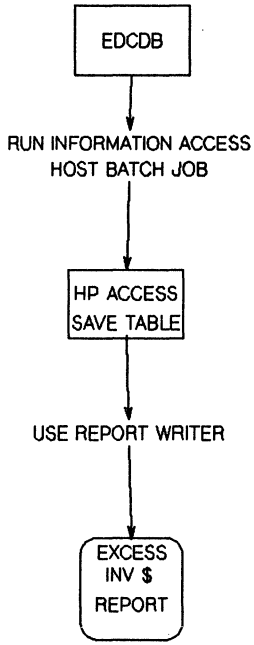
To fully automate the process of generating the excess inventory dollars report, a new feature of Information Access, the Host Batch Facility, was used. The Host Batch Facility provides a way to perform a series of database manipulations and save the resulting data for subsequent access. These batch jobs can be run on the PC, or if the PC user does not want to tie up their PC during execution of the batch job, the job can be run on the HP 3000. The controllers wanted to check on excess inventory dollars once a month so a batch job was created to run on the HP 3000, at a scheduled time. This batch job is shown on the following page:

```
!JOB HDPJOB,MGR.HPOFFICE
!COMMENT ****************************************
!COMMENT *  PRODUCES EXCESS INVENTORY DOLLARS BY *
!COMMENT *  CONTROLLER TABLE                    *
!COMMENT ****************************************
!COMMENT
!RUN HDPBATCH.PPC.SYS
REMOTE PURCH BUY
* **** DELETE LAST MONTH'S SAVED TABLES ****
SELECT "EXCESS-TEMP"
DEL_TAL
RESUME
SELECT "EXCESS-REPT"
DEL_TAB
RESUME
* **** LOAD VIEW TABLE WITH WEEKS-OF-SUPPLY > 52 ****
SELECT EXCESS-FINAL
OUTPUT_TAB
SELECT HPACCESS
PERFORM_OUTPUT "EXCESS-TEMP"
* **** PARE DOWN FOR SPECIFIED CONTROLLERS ****
Q_RESULT
SEARCH " (((CTLR >= ' 1') AND (CTLR <= '19')) OR (CTLR = '27') OR &
(CTLR = '30') OR (CTLR = '32')) "
PERFORM_Q
* **** SAVE TABLE FOR REPORT ****
OUTPUT_TAB
SELECT HPACCESS
PERFORM_OUTPUT "EXCESS-REPT"
EXIT
!EOJ
```

With the use of the above batch job, Information Access has decreased the time required to generate the excess inventory dollars report from several hours to five minutes per month. Shown below is the process flow after implementing Information Access.

PROCESS FLOW FOR EXCESS INVENTORY $ REPORT

(AFTER USING INFORMATION ACCESS)



An added benefit to this project was that once the controllers became familiar with Information Access and how powerful it is, they began using the product interactively to aid in decision-making and for generating small ad hoc reports. Some of the additional functions which are performed with Information Access by the controllers are listed below:

- Retrieve detail information for a particular vendor or assembly. Datasets ITEM-STRUCTURE and ITEM-DATA are joined on PART-NUMBER. The items retrieved include STD-COST, PARENT, and COMPONENT PARTS.
- Determine all open orders for parts for various assemblies. Datasets ITEM-STRUCTURE and PURCH-ORDER-DETAIL are joined on PART-NUMBER. The items retrieved include ORDER-STATUS, DATE-DUE, and PARENT-PART.
- Cost analysis to determine whether a board should be scrapped. PARENT-PART, QUANTITY-PER, and STD-COST are retrieved and output to a spreadsheet for 'what-if' analysis.

- Provide detail for PROMIS-generated graphs that report delivery performance of a supplier. The graphs report only on-time, early, and late delivery. Information Access was used to supply further detail such as order number, date due, and date received.

As can be seen from this example, Manufacturing benefited not only from the time reduction in generating their excess inventory dollar report, but also from the fact that the controllers felt comfortable enough with Information Access to use the product across all aspects of their jobs which required data retrieval.

EXAMPLE - INFORMATION ACCESS AND SDN GENERATION

Introduction

When a product goes through a new release or update, generally there are materials which must be sent to customers on particular subscription services. Product Support is the group responsible for determining what materials go to which services, and communicating this information to the Software Distribution Center via a Software Distribution Notification (SDN).

In this particular case, an update to Information Access was released on the UB-Delta-3 MIT, which required a distribution of new manuals for both HP Access Central and HP Access, as well as new PC software. Because the product structure had been modified with the UB-Delta-1 release (from HP Access Central and HP Access being purchased independently, to being purchased as one product called Information Access), the support services were going through a transition. We had customers on support services for the original product structure, and we had customers on the support services for the revised product structure. At final count, there were 25 customer subscription services which required SDNs in order to trigger SDC to make a distribution to the customers. In addition, two other items were required for each subscription service. First, a cover letter, which would be sent with the actual distribution describing the contents of the package, and second, an HP internal document which would track the actual material cost associated with the distribution to a particular service.

On the average, a product update may require five SDNs, requiring a half day to complete, which includes determining what materials need to be included in the distribution, getting appropriate part number information from manufacturing, writing the SDN, and writing the covers. To complete the work for the 25 SDNs, plus determine how to track material costs for each subscription service was estimated to take one week. Although, this doesn't seem like much, when you consider that Product Support does this for every update or new release all of our products requiring a distribution, there certainly needed to be a more efficient way of handling this process and associated paperwork. Information Access, and especially Report Writer, provided the solution.

Database Design

A simple IMAGE database was designed to serve as a central repository for all of the data required. The database has the following schema:

```
BEGIN DATA BASE SDN;

PASSWORDS:
 1  PASS;          << SANDY >>

ITEMS:
 DESCRIPTION,     X30 (1/1);        << PART-NUMBER DESCRIPTION >>
 MATL-ID,         X6 (1/1);         << PART-NUMBER MATERIAL ID >>
 PART-NUMBER,     X12 (1/1);        << PART-NUMBER >>
```

```
 PRODUCT#,          X10 (1/1);          << PRODUCT-NUMBER >>
 QTY,               I (1/1);            << PART QUANTITY >>
 SMS#,              X14 (1/1);          << SMS PRODUCT NUMBER >>
 SMS#-DESCR,        X36 (1/1);          << SMS PRODUCT NUMBER DESCRIPTION >>
 SMS#-QTY,          I (1/1);            << QUANTITY ON PARTICULAR SMS >>
 UNIT-COST,         R2 (1/1);           << UNIT-COST OF A PART >>

SETS:
 NAME:              SUBSC-SVCS,MANUAL (1/1);
 ENTRY:             SMS#(1),
                    SMS#-DESCR,
                    PRODUCT#,
                    SMS#-QTY;
 CAPACITY:          50;

 NAME:              PART-MASTER,MANUAL (1/1);
 ENTRY:             PART-NUMBER(1),
                    MATL-ID,
                    DESCRIPTION,
                    UNIT-COST;
 CAPACITY:          100;

 NAME:              SUBSC-PARTS,DETAIL (1/1);
 ENTRY:             SMS#(!SUBSC-SVCS),
                    PART-NUMBER(PART-MASTER),
                    QTY;
 CAPACITY:          500;

END.
```

The factor taken into account with this database was to allow for rapid retrieval of the SMS# and the PART-NUMBER, since most queries would be done on these data items. This was accomplished by both of these items being search items.

Planning The Configuration

In planning the configuration, Worksheet 3 was the only worksheet completed. Worksheets 1 and 2 were not completed because it was already known that one user would be using the configured tables and that all three IMAGE datasets were required, which is the information the Worksheets 1 and 2 would have provided. Worksheet 3 was helpful because it provided the opportunity to plan the view table definitions prior to running the Administrator Utility. Completed Worksheets for the two required view tables are shown on the following pages.

CONFIGURATION WORKSHEET 3 **VIEW TABLE DEFINITION**

TABLE NAME: SDN-INFO DATE: 05/87

DESCRIPTION (optional): THIS TABLE WILL PROVIDE ALL INFO FOR GENERATING SDNS

-- ITEMS --

TABLE NAME	.	ITEM NAME	=	EXPRESSION	,
SUBSC-SVCS	·	SMS#			
		SMS#-DESCR			
		PRODUCT#			
PART-MASTER	·	PART-NUMBER			
		MATL-ID			
		DESCRIPTION			
		QTY			

-- USING CLAUSE --

((SUBSC-SVCS<SMS#> JOIN SUBSC-PARTS<SMS#>)<PART-NUMBER> JOIN PART-MASTER<PART-NUMBER>)

-- WHERE CLAUSE --	-- SORT CLAUSE --
	SUBSC-SVCS A, PART-NUMBER A

NOTES:

CONFIGURATION WORKSHEET 3				VIEW TABLE DEFINITION
TABLE NAME: COST-INFO				DATE: 05/87
DESCRIPTION (optional): CONTAINS INFO ON DISTRIBUTION COSTS PER SMS#				
-- ITEMS --				
TABLE NAME	.	ITEM NAME	=	EXPRESSION
SUBSC-SVCS PART-MASTER	. .	SMS# SMS#-DESCR PART-NUMBER DESCRIPTION UNIT-COST QTY SMS#-QTY		
-- USING CLAUSE --				
((SUBSC-SVCS<SMS#> JOIN SUBSC-PARTS<SMS#>)<PART-NUMBER> JOIN PART-MASTER<PART-NUMBER>)				
-- WHERE CLAUSE --				-- SORT CLAUSE --
				SUBSC-SVCS A, PART-NUMBER A
NOTES:				

Configuration Of Information Access

Configuration of Information was a very straightforward process. From the Administrator Utility, the following steps were completed:

- The database, SDN, was configured.
- The three IMAGE datasets, SUBSC-SVCS, PART-MASTER, and SUBSC-PARTS, were configured as IMAGE tables, with all data items from each dataset included.
- Using the two worksheets shown above, two view tables were configured.
- An existing access group, SUPPORT, was assigned to the five tables to provide table security.

Using The Report Writer

Once the data was available to be accessed, the one task remaining was to design the reports to be used. The first of these report formats, the actual SDN form, is shown below. The highlighted fields indicate which fields are actually drawn from the HP Access table. The remaining text is simply a permanent part of the form itself.

SOFTWARE DISTRIBUTION NOTIFICATION

DISTRIBUTION DESCRIPTION: SMS#-DESCR
IS THIS PART OF A LARGER DISTRIBUTION ? PART OF DATE: DATE

PART INFORMATION:

ER DIV	PART NUMBER	MAT'L ID	PART DESCRIPTION (30 CHAR)	QTY	SUPPLY METHOD
D500	PART-NUMBER	MATL-ID	DESCRIPTION	QTY	MAST

FOR DISTRIBUTION:
[X] US [X] CANADA [X] OTHER ICON [X] HPSA [X] SPECIAL

PRODUCT INFORMATION:

PRODUCT NO OPTIONS PRODUCT NO OPTIONS PRODUCT NO OPTIONS
PRODUCT#

IS SDC SUPPLYING DIVISION FOR THE PRODUCT OR PART? YES [X] NO [X]
IS THIS A NEW PRODUCT? YES [ ] NO [X]

SERVICE INFORMATION:
(LIST INTERNAL AND OTHER SERVICES MAINTAINED BY SDC)
SMS#

COMMENTS/PRINT INSTRUCTIONS/SPECIAL ACTIONS ETC.
THIS DISTRIBUTION TO BE COORDINATED WITH THE UB-DELTA-3 DISTRIBUTION.

WHAT TYPE OF ORDER? HEART [ ] I.O. [ ]

JOINT PURCHASE VENDOR:

DIVISION INITIATOR : SANDY LYNCH

SEND THIS FORM TO THE SOFTWARE DISTRIBUTION CENTER.

The second report, which tracks the distribution cost for a particular subscription service, takes advantage of some additional Report Writer features. Note the following calculated fields in the report layout shown below, MATERIAL COST OF THIS DISTRIBUTION and TOTAL BILLING TO OSD.

```
SMS# SMS#                    SMS#-DESCR

PART INFORMATION:
                                          UNIT                TOTAL
PART NUMBER   DESCRIPTION                 COST        QTY      COST

PART-NUMBER   DESCRIPTION                 UNIT-COST   QTY      UNIT-COST*QTY


      MATERIAL COST OF THIS DISTRIBUTION        SUM-OF-TOTAL-COST

   TOTAL BILLING TO OSD                 SUM-OF-TOTAL-COST*SMS#-QTY
```

Summary

Information Access is a product which can be used to provide productivity imporvements across all functional areas of an entity. Certainly, planning is required to fully implement the product, although some simple worksheets make the actual configuration of Information Access very straightforward. By taking advantage of the functions available with view table definitions, such as the SUB function, CASE statements, JOIN, and Where clause, you are able to retrieve information which may not have been available before automatically. By also making use of Report Writer, you are able to create reports for presenting the data retrieved in an easy to read format, displaying just the information required. One last important point is that once you provide people with Information Access to perform a process, they will most certainly find additional ways to use the product in additional day to day tasks.

Using C on the HP3000

Paul Hays and John MacNaughton
COGNOS INCORPORATED
3755 Riverside Drive
Ottawa, Ontario
Canada
K1G 3N3

Abstract

On most mini-computers, the C programming language is becoming very popular with both programmers and system administrators. While this increased popularity is due in part to the micro market, the power of C is very hard to argue with. Since C is available on the new 840 series and, we hope, on the 930, it could become more prevalent on the HP3000.

The architecture of the HP3000 creates some interesting and rather unique problems for someone trying to develop a large system in C. Even a mid-sized system from other hardware may run into numerous brick walls trying to run on the 3000. Despite these difficulties, you may want to consider using C to develop new applications. This paper will discuss some of the pitfalls of using a fully ANSI standard C compiler on an HP3000 for system development.

Table of Contents

1. Introduction 3
2. The C bible 4
3. The ANSI committee 4
4. Why use C ? 5
4.1. The Manager 5
4.2. The Programmer 6
4.3. Why not use C ? 7
5. Possible problems using C 8
5.1. File System 8
5.2. Environment 9
5.3. Memory 9
5.4. Procedure Calls 10
6. Hard-to-use features of the HP3000 10
6.1. PB space 10
6.2. USLs 11
7. Recommendations 11
8. Acknowledgement 12
9. Bibliography 13

1. Introduction

This paper examines some of the benefits and problems associated with using 'ANSI compatible' C on the traditional HP3000. It is not a critique of any of the C compilers now available for the HP3000, nor does it recommend any particular compiler. It should, we hope, provide you with enough information to allow you to determine if C is the development language for your applications and, if so, which compiler is best suited to you.

A point worth remembering is that although C implementations on the HP3000 are still in their infancy, a wealth of expertise gained from other environments will be applied to their development. There are also many portable C programs in the outside world that have been just waiting for a good C implementation to appear on your computer. Expect C to mature rapidly on the HP3000.

There are at least three sources of programs that you may consider for C:

- a translation of an existing HP3000 application
- a new application
- an existing C application designed for a machine other than the HP3000

Each of these possible sources of new code demands very different criteria when trying to decide if a given C implementation is suitable, or if C is even the language to use.

It is a perverse fact that any significant program will almost always produce different results than the original when rewritten, regardless of the target language. No programmer can resist the temptation to 'improve' a program while rewriting it, and any program has undocumented behavior that will change. This argues that you should have the best possible statement of the requirements for your program before the rewrite begins.

C provides a good range of features that you can substitute for source language features. The flexibility can be hazardous, but this may be preferable to the awkward restrictions that are imposed by some other languages. Pascal, for example, is very restrictive, since it was designed for teaching rather than system programming. C cannot, of course, emulate every feature of your source language. Like Pascal and Spl, C descended from Algol; these "block-structured" languages share common concepts that should simplify translation. (Don't translate your Apl application to C.) When deciding what language to convert to, consider similarities between the source and target, and ensure that the target language offers suitable constructs.

If your application (a relational database, for example) uses memory extravagantly, then major surgery may be needed to make it fit into the HP3000's space limitations. Ensure that the application is small enough to fit the 60 Kilobyte data space available on the HP3000, or that it can be suitably restructured to fit. By the way, major surgery is occasionally fatal!

Building a new application in C is easier than translation in at least one respect, since the features of only one language need be considered. Writing a new application is the best way to learn if C on the HP3000 can do what you need, because there should be few preconceived notions about the details of implementation.

For either of the above cases, it is necessary to decide whether to use the i/o library functions supplied with your C compiler, or to perform i/o using the underlying MPE intrinsics directly. The decision is influenced by at least two factors; portability and performance. If your program relies on MPE intrinsics, then it can only run under MPE, or, possibly with some modification, under MPE/XL on a series 900. The same is also true if you require any filesystem features that are not described in the ANSI standard, such as job/session temporary files, even if your program uses extended operations available in

the C i/o library. Portability is a matter of degrees, and is seldom absolute.

The easiest task for a developer, and the best test of your 'ANSI compatible' C environment, is to port a C application from another system. We have found that this can be trivial. It is gratifying to see programs that run on DEC and Sun3 machines compile and run on the HP3000 with little or no modification.

2. The C bible

If you look in the bookshelves of most C programmers, you will find a relatively small book entitled "The C Programming Language" by Kernighan and Ritchie from Bell Labs. For those used to manual racks full of reference material, this thin soft cover book may be a surprise. This book contains the first published definition of the language C, and most C compilers follow the book faithfully.

There are at least three areas where the definition proves inadequate, however. Firstly, the writers made no attempt to define i/o statements, nor did they thoroughly describe the i/o functions in the runtime library. Secondly, due to the number implementations of the language in varied environments, many extensions have evolved. A given desired result may require different syntax under different implementations. Thirdly, some of the definitions in the book leave too much room for interpretation. This oversight has allowed compiler writers to create slightly differing semantics for the same syntax. Luckily, this is seldom a large problem, as long as the programmer is aware of the dangers of using subtle coding practices. This last point is true for most languages; for example, you really shouldn't count on the order of evaluation of procedure arguments.

This book, known as K&R for convenience, attempts to define the philosophy behind the C language definition. This, more than anything, has fostered reasonably similar implementations on different computers. The authors were also very frank about some of the missing capabilities in their definition of C, such as parallel operation, and the possibility that some of the operators have the wrong precedence. The writers also describe how some of the syntax has changed to make C less ambiguous. All of this shows the aura around the C language to be friendly, open, and very dynamic.

3. The ANSI committee

In the summer of 1983, the American National Standards Institute convened committee X3J11 to produce a standard for the programming language C. The project has had support from a broad cross section of the computer industry, including both Hewlett Packard and Cognos.

The major thrust of this project has been to "provide an unambiguous and machine-independent definition of the language C", according to the SPARC document 83-079. This standard is intended to clarify the syntax and semantics of C, as well as determine what restrictions and limits could be imposed upon it by a conforming implementation of the language. To define the i/o facilities and other features of a practical language that K&R left incomplete, the committee adapted a widely-used set of library functions from the /usr/group committee of Santa Clara, California.

The C standard does not specify implementation methods. It simply states what language constructs have to be accepted, and the meaning of what is accepted. The document defines three classes of behavior for a conforming system at any point: unspecified, undefined, and implementation-defined behaviors. If the behavior is unspecified, the standard imposes no requirements on a correct program construct. In compiling a procedure call, for example, the standard permits the implementation to push arguments onto a stack in whatever order works best in the target environment, leaving this behavior

unspecified. (Expect a compiler for the HP3000 to build procedure calls in such a manner that C functions can call or be called by Spl routines or system intrinsics.) If the behavior is undefined, the standard imposes no requirements on an erroneous program construct. One example of undefined behavior is the result of an attempt to divide a number by zero. (Programmers generally attempt to avoid operations that have undefined behavior, with varying success.) Behavior is implementation-defined if the standard imposes no requirements on the compiler beyond documenting the behavior. A conforming compiler must come with documentation describing, for example, the allocation of any padding used to align members in a data structure.

The ANSI standard is defined so that extensions to C may be accepted by the compiler as long as two conditions are met. The first condition is that any strictly conforming program must behave correctly. (A strictly conforming program is one that uses no code construct that is not explicitly defined by the standard.) The second condition is that all implementation-defined characteristics and extensions to the standard are documented.

The ANSI committee expects the proposed C standard to be accepted sometime in 1987. This causes some problems in timing, since some of the more advanced features proposed in the standard may take some time to be implemented in the compilers. There are no secrets in the standard, however, so most compiler writers have already started implementing those features.

4. Why use C ?

4.1. The Manager

We feel that the DP manager's main concern with programming languages is their affect upon the productivity of the programming staff. The manager wants to equip the staff with adequate tools and ensure that programmers are happy with the tools. One of the things that can make programmers happy is letting them work in a language that they like. C may well be the most popular programming language in the world, aside from questions about whether it is the world's best one.

While Cobol is probably the most extensively used language on the HP3000, there are types of applications that Cobol just isn't suited to. Until recently, Spl has been the only other choice if your program has to manipulate bit fields or use the full memory capabilities of the HP3000. (Perhaps you will agree with us that large applications are difficult to build in Fortran; straightforward numerical applications are its forte.)

Spl is seldom taught at the university or college level, making it difficult to find experienced Spl programmers. Unless you can raid other HP shops, you will have to invest the time to train them yourself. On the other hand, C is widely available for the home computer market on micros, and is taught at many colleges and universities. The available pool of programmers also includes the large world of IBM and DEC computers, to name a couple.

One major advantage of C over Spl is the availability of generalized data structures that can be used to make the intentions of a programmer more clear. The end result of using a more expressive language is code that can be less expensive to build and, more important, to maintain throughout its life cycle.

Another advantage of C is that the language is familiar to a large community of compiler writers who understand the techniques of optimization. If your C compiler offers code optimization, look for improvements in processor time in programs rewritten from Spl to C; the Spl compiler seems to perform little optimization.

Also available to the DP manager is a whole world of productivity software that could be adapted to fit local requirements. These include editors and code control software, as well as an awesome array of user-written tools in C. There are many C products existing on other computers for which translation to Spl for the HP market would not be cost effective. With a competent C environment available, the effort needed to transplant these packages onto the HP3000 becomes minor; software vendors may now migrate their programs to the HP3000. To be honest, probably the first machines to get these products will be the new series 800 and 900 Spectrum machines, since they don't suffer from the older HP3000's restricted data space. HP3000 owners should see some packages on their machines too, though.

4.2. The Programmer

Like Spl, C is a low level/high level language. This means that, when required, the programmer can get right into the bits of the machine. Most of the time this really isn't necessary because the compiler will handle the details for you. Most, if not, all modern high-level data structuring techniques are available in C, as are the usual elements of structured flow-control. The biggest advantage C has over Spl is the addition of real data structuring capabilities.

Another improvement that can help clarify larger applications is C's support for definition of global data in any separately compiled source module. This feature permits you to structure the application cleanly, since any sub-application can define its own global data independently of the main application. By a sub-application, we mean a set of related functions that may be implemented in one or several modules. Global data may be required solely for the internal operation of the sub-application. The file table maintained by the i/o functions of a C runtime library is a good example of this practice.

The macro facility of C permits the flexible use of parameters to produce clear and efficient in-line code for small repetitive operations. Judicious use of macros can greatly improve readability of the code, and save typing too.

The ANSI standard library provides a powerful and well-documented set of fundamental procedures for

- a full complement of math operations
- string handling
- flexible and portable i/o
- powerful text formatting functions
- dynamic space allocation
- random numbers
- date and time conversions

Using the standard functions not only simplifies the programmer's task, but improves maintainability because the common library is well known to all C programmers everywhere.

Symbolic debuggers can save hours of tedious work by automating what has until now been largely manual labor for HP3000 programmers. At least one of the C implementations available for the HP3000 includes a nifty symbolic debugger with the compiler. Similar tools have long been standard on other machines, and one is included with the operating system on the Spectrum series 800. We sincerely hope that HP will give us such productivity tools on the series 900, too.

Let's suppose you use a fourth generation package and wish to dynamically load functions that are coded in C. No problem! If the call frame generated by the 4GL list is acceptable to an Spl procedure, your C function should work just fine, assuming the compiler generates a standard call frame. Of course, your C compiler must be able to write to a USL or provide some other means to move the binary into an RL or SL, so that the 4GL package can load the resulting function.

If you find it necessary to apply patches to program files as part of your business, check into the kinds of listing files that the compiler can produce. Note that the better the compiler's optimizer functions are, the less familiar the assembly code looks. Patching is far easier if the listing shows which assembler instructions are associated with each statement of the high-level language.

The K&R "bible" states that the definition of C is small, which should mean that the basics of the language are easy to learn. Any language can be used to perpetrate guess-what-this-does programs, and C is no exception. The same powerful structures that make it possible to write clear, concise, and maintainable code to perform complex operations can also be used to make incredibly obscure and costly code. Any truly professional programmer realizes that programming languages are intended to communicate not just with a compiler, but also with an audience of maintainers over the life cycle of a product.

4.3. Why not use C ?

There are thorns in the bed of roses, however.

In this language, what you see is what you get. If you try to pass an integer pointer to a routine that expects an integer, then that is exactly what the compiler will do to you. Spl offers some built-in protection against this kind of accident by forcing you to use type conversion functions. While it can lead to some rather subtle and hard-to-find bugs, this kind of flexibility is also a source of C's power. (The LINT test compiler available on UNIX systems was written to try to solve this problem by monitoring such points of syntactic weakness. A similar tool would be a welcome addition to other C environments.) Some compilers are better than others at finding coding errors, but the level of checking falls into the area ANSI has declared to be implementation-dependent.

The ANSI committee has proposed a solution to this whole problem with arguments and parameters by specifying 'function prototypes'. Function prototypes are statements that serve a purpose very similar to the intrinsic definitions that HP3000 programmers are used to. Among other things, a prototype can indicate to the compiler that the type of an argument passed to a function is to be implicitly converted to the type expected by the callee. Few compiler writers have implemented function prototypes yet, but most are in the process. When they are available, function prototypes will eliminate some obscure problems caused by type mismatches when calling external procedures.

Arithmetic expressions in C, unlike those in Spl, are permitted to contain mismatched data types. C provides cast operations, (similar to Spl's type transfer functions), that can make the programmer's intention clear. Since casting is optional, the language provides a set of rules to determine the default cast operations if none are specified. In its proposed C standard the ANSI committee has set a trap for the unwary by changing the rules slightly from those used by many existing compilers. Although the new rules are more intuitive than the old ones, they may well break code that relies upon the default casting rules.

The structure of argument lists on the HP3000 might cause trouble for code imported from other machines. C compilers on many machines allow trailing arguments to be dropped when calling any function. Such code works if the callee doesn't access the corresponding formal parameters. In the conventional call frame on the HP3000, the right-most argument is supposed to live at address Q-4, the next at Q-5 etc. Now, if you call a function passing three arguments (1, 2, 3) when the function defines five parameters (a, b, c, d, e), then references in the called function to a or b access general junk. As the ANSI standard puts it, the result is undefined. In Spl, the "option variable" construct handles the problem in a straightforward (but somewhat expensive) manner. The ANSI standard specifies a solution for C which requires that a special function prototype must be declared when the caller is compiled, and that the callee must access its parameters indirectly via a macro named "va_arg". The mechanism

should be easy enough to use, but is a bit of a nuisance if you are used to the Spl convention. Assuming that your compiler doesn't yet implement the new function prototype regime proposed by the ANSI committee, you should ensure that it provides a workaround for this situation.

If you use many "option variable" procedures in Spl code to be translated, then you may encounter another difference in C argument lists. Unlike Spl, C does not permit arguments to be dropped from the middle of an argument list. Often the solution is to insert a default value into the argument list wherever the offending function is called. This solution doesn't work if the callee behaves differently when the argument is missing than it does when the argument has any of its possible values. ("Missingness" can be treated in Spl as one of the possible values of a parameter.)

If you intend to use C for general programming that requires calls to MPE intrinsics, you should investigate the facilities available with the compiler to deal with Spl-style variable argument lists and missing arguments. In particular, it is useful to extend the language so that arguments can be omitted to obtain default values for certain intrinsic parameters.

5. Possible problems using C

Doing development in any language new to you or your computer is always an interesting experience. If you decide to switch some of your development to the language C, it will be no different. However, in this case you acquire two sets of problems: a language possibly new to your programmers, and compilers new to the HP3000. There is the some possibility that the senior technical programmers have more experience on the HP3000 than the developers of the C compilers. This means that your staff may be more aware of the intimate details of the HP3000 than the compilers are. This may lead to some confusion, where some "strange" action does not seem natural to your experienced staff and they don't know if it's a problem with their understanding of C, or a problem with C not doing the natural thing for the HP3000.

5.1. File System

The first problem you will encounter when you import a program from a different machine may be a small matter caused by differences in the filesystem. By convention, the names of files referenced by the "#include" statement often contain a portion called an 'extension', usually ".h". On the HP3000, the part following the '.' specifies a file group. The ANSI standard necessarily leaves the method used to determine the location of the file up to the implementors of the compiler. Likely places to look for a header file like "mydefs.h" might be in the current group (strip off the extension), in a group named "h" in the current account, or in the group containing the source file (which might not be your current group). Of course, you should be able to use :file equations, but you must know whether you need an equation for "mydefs" or for "mydefs.h". To further complicate matters, the "#include" statement has many possible options which could alter the search path used by the compiler to find the file. The compiler writers could help us out by providing a means to supply the compiler with a list of groups to search in sequence for the include files. In our formal development environment, all source files reside in groups whose names contain a revision number. It's no good for the compiler to look for our header files in group "h" if they must be in group "h506".

The next problem you may stumble across is the i/o library provided with the C compiler. C i/o is generally character-oriented rather than line- or record-oriented. Normally, text files to a C program are streams of characters delineated by the <new-line> character. Text files to the HP3000 are generally fixed length files with or without carriage control, so the i/o routines must perform some conversions. (One type of text file that we haven't been able to access using C i/o is a compact and efficient format

used by the software tools from the Robelle company.)

Standard C also has no notion the HP3000 implementation of the job/session temporary file. There are standard library functions to determine unique file names so that C programs can build files that won't conflict with existing files. Note that a file opened with such a 'temporary' file name will still appear in the permanent file domain when it is closed.

The whole HP3000 file system is essentially opened up by the myriad of options available on the FOPEN intrinsic. A vendor of C for the HP3000 should be expected to extend the C i/o library to handle most or all of the features of MPE's FOPEN intrinsic. This is necessary so that the rest of the powerful i/o operations in the library are available with all types of MPE files.

5.2. Environment

Interactive C programs, particularly text editors, often include some means by which the user can 'escape' to the operating system's command interpreter to issue a command. To support this, the ANSI C standard specifies that the runtime library must contain a function named 'system' which is intended to communicate with a command interpreter in the host environment. The committee has also stated that this function "allows a program to suspend its execution temporarily in order to run another program to completion". Unfortunately, HP does not provide a means to 'programatically' call MPE's command interpreter to run programs or execute user-defined commands. Several third party products on the HP3000, however, include command interpreters to handle these requests. Ideally, C implementations for the HP3000 would include such a command interpreter to support interactive programs. In any case, the users of such programs should be aware of any limitations imposed by the implementation of 'system'.

Early versions of MPE had limited capacity to pass run time parameters to a program. That capacity was extended with the "info" string, although there are still some size limitations. The ANSI standard specifies a mechanism by which C programs receive parameters from the operating system in a parameter of the main function. It is the responsibility of startup code from the C library to format the info string into this form. We note that MPE still provides the value of the PARM option of the command RUN to user programs. C on the HP3000 should provide some mechanism to allow access to this parameter, because by the time your main procedure starts it becomes very difficult to get at the memory location where MPE has stored this value. We recommend that the standard function 'getenv' be adapted to this requirement.

5.3. Memory

The ANSI proposal requires that internal identifiers can be up to 31 bytes each, with external identifiers 6 long. It seems obvious that, unless the C compiler does something fancy, the compiler itself will run out of symbol space. (The Spl compiler only pays attention to the first 15 bytes of a name, and it occasionally runs out of space.) If the C compiler is to be used for large applications, we recommend that the compiler provide a means by which the user can control the number of significant characters used for symbols.

When running a program that has very dynamic memory demands (for example, a generalized report writer), memory space can present a major problem on the HP3000. This means it is up to the running program to do its own memory management and "garbage collection". With the restricted memory space available on the HP3000, memory fragmentation can be disastrous. After many cycles of allocating and freeing blocks of various sizes, fragmentation can cause the heap manager to report an

allocation failure even though there may be plenty of memory available. We feel that the developers of any heap manager for the HP3000 should thoroughly document how their memory allocation routines work and take steps to avoid causing fragmentation in all of their code.

Once again proving the adage that you can't get something for nothing, using the C i/o routines will probably cost you memory as well as speed. This is because to allow all the nice functionality that C i/o has, the routines must have their own buffers. This means there will normally be two buffers active, yours and the library's (plus MPE's of course, but that is there most of the time, anyway). We would like to see standard i/o routines that use nobuf access wherever possible.

5.4. Procedure Calls

The HP3000 implements in microcode many operations that are usually performed via calls in C environments. On our series 48, a procedure call to a memory-resident segment takes on the order of 25 microseconds, but to a non-resident segment it can take milliseconds. Let's examine some initialization code, for example. In Spl, you can write

```
MOVE byte'array := (10)" ";
```

which moves ten spaces into some byte array. You could also code

```
MOVE byte'array := "          ";
```

which will have the same affect. Neither of these statements causes a call; they invoke efficient microcode. In C, you have a choice of run time library functions to handle the same requirement:

```
strcpy (byte_array, "          ")      or
memset (byte_array, ' ', 10)
```

(Actually, the strcpy doesn't do quite the same thing. It appends a nul character to the target array because, in C, strings should be nul-terminated).

This performance hit could be painful for data-processing applications that initialize and copy blocks of text frequently. There is nothing in the ANSI standard to prevent C compilers from providing built-in procedures that implement the simple array handling routines as inline code. To take advantage of some rather basic architecture in the HP3000, C vendors should examine this issue closely and ensure that the implementation makes effective use of the instruction set.

On the subject of efficiency, note that a typical implementation of a C runtime library uses a macro (named putc) to make character-by-character output more efficient than if a PCAL were used for each character moved. You might try a metered run of a test program that writes some large number of characters to ensure that the implementation is efficient.

6. Hard-to-use features of the HP3000

This is by no means a complete list but should mention the most difficult features of the HP3000 for an ANSI-conforming C compiler to use.

6.1. PB space

The ANSI proposal includes a storage attribute of 'const', which is an indication to the compiler that

the value of a variable may not be changed either directly or indirectly. Since the HP3000 offers no access protection for your data stack (if you can read it, you can write it), the restrictions needed for const storage can be enforced at compile time only. The const attribute could at first glance be interpreted to be the same as Spl's PB attribute. Unfortunately, the HP3000 architecture imposes restrictions on PB storage that are not imposed on ANSI's const storage. Const variables can be used in the same manner as any other data except that they can't be the target of any sort of assignment. Spl data that have the PB attribute can only be used as the source of an assignment or move instruction, or the constant in a string comparison. (As noted elsewhere, C doesn't normally do string comparisons inline so the compilers don't use PB space for this. They could, though.) Also, data to be accessed via the machine's P register must reside in the same code segment as the current executing instruction. This is why Spl restricts its use to local arrays. There is no equivalent to this special storage mechanism in C.

There are critical uses of PB space on the HP3000, however. We use it to store large amounts of seldom-needed text (for example, error messages) for infrequent retrieval. It is impractical to keep the error text in the limited storage area of the data stack; this little-used data should remain out of main memory until needed. If the C compiler doesn't allow access to PB storage, then this sort of design cannot work in C. In order to make this important resource of the HP3000 available to users of C, the language should be extended slightly to add a const_PB storage type, and to restrict its use in the same manner that Spl restricts it. An alternative possibility is to permit a C program to call an Spl procedure which can copy information from a local PB area back to the caller. Such a mixed-language scheme might be acceptable if its use did not impose any further restrictions on the non-Spl modules of the program.

6.2. USLs

A compiler design which, like HP's compilers for the HP3000, creates relocatable modules in a standard USL file would seem straightforward. This approach allows the inclusion of other languages in the program, and the use of PREP, insulating the compiler from changes to either USL formats or program files. (HP is not likely to make such changes.) It naturally assumes that the USL structure and the PREP program are capable of handling all requirements of ANSI C, if the implementation is to conform to the ANSI standard. One case where we believe that the USL structure is not adequate for C is the requirement that only the module which contains the program's entry point, i.e. the 'outer block', may define global data. A design that offers no relief from this restriction does not conform to the ANSI standard since ANSI C permits any module in a program to define global data. Whether such a restriction is important to your application is for you to decide; you should be aware that applications that you wish to import from other machines may require modification to meet the restriction.

It would seem that a new linker program may be required in order to support full ANSI C on the HP3000. We believe that any linker program supplied for the HP3000 should be capable of resolving references from a standard USL file as well as any new type of relocatable object created by the C compiler. If this criterion is not met, some programs which otherwise conform to the standard could not be linked with modules written in the other HP3000 languages.

7. Recommendations

In the same way that Spl is not for everyone, C is not a global solution for all problems that plague the development shop. If you want to tap a resource bank of C applications or move your C application into the HP3000, C is worthy. If your applications are in Spl and maintenance costs are high, or if maintaining two sources will be too costly when the Spectrum arrives, then we feel that there may be benefits in moving them to C. One point is that HP is not going to provide a native-mode compiler for

the Spectrum machines (there is a compiler available from a third party), but C is available on the series 800 and it is expected to be on the series 900. Of course, if you are currently using a series 48 with no plans to upgrade in the next five years, then that really doesn't matter. In any event, Spl seems unlikely to become the language of the future.

Any or all of the possible problems mentioned in using C may be solved quite properly by the C compilers available for the HP3000. As noted, some of the problems only occur in large applications or those which must mimic specific actions in other languages.

We suggest trying a couple of small, straightforward applications in C so that your technical staff can determine for themselves the capabilities of the language and of its implementations on the HP3000.

If you intend to port a C application to the HP3000, you will need to analyze the application and the compiler to ensure that the C implementation is sufficiently compatible and that the memory usage is within the HP3000's limits. If you intend to migrate a product to Spectrum native mode in C, you need an i/o library with a highly compatible set of extensions to support the MPE filesystem and/or a clean compiler interface to the MPE system intrinsics. For large applications, you may need a workaround to enable use of PB data.

We hope that this paper has clarified some of the issues that you need to consider before deciding to use C and when selecting a C implementation for your organization.

8. Acknowledgement

The authors would like to thank Tim Chase and Bruce Frank of Corporate Computer Systems, as well as Dr. Jay Anderson and Chuck Stern of Tymlabs Corporation for their kind cooperation.

DEC is a trademark of Digital Equipment Corporation.

HP, HP3000, and Spl are trademarks of Hewlett-Packard Company.

Sun3 is a trademark of Sun Microsystems.

UNIX is a trademark of AT&T in the United States of America and other countries.

9. Bibliography

The C Programming Language, Brian W. Kernighan and Dennis M. Ritchie, (Prentice-Hall, 1978)

Draft Proposed American National Standard for Information Systems - Programming Language C, X3 Secretariat, Computer and Business Equipment Manufacturers Association, 311 First Street N.W., Suite 500, Washington, D.C. 20001-2178

Rationale for Draft Proposed American National Standard for Information Systems - Programming Language C, X3 Secretariat, Computer and Business Equipment Manufacturers Association

Using C for Migrating to HP's Precision Architecture, Tim Chase and Chris Maitz, Interact Magazine, March, 1987, p 35.

HP3000 Computer System Systems Programming Language Reference Manual, Hewlett Packard.

COPING WITH CHANGE

or

Help! A New Release is Coming

Albert L. Magid, President
ALDON Computer Group
405 14th Street, Suite 715
Oakland, CA 94612

COPING WITH CHANGE

or

Help! A New Release is Coming

In the world of data processing, change is Inevitable. Anyone with any experience working in the field of information handling knows that change is the rule. The only unknowns are the rate of change and the magnitude of the changes that will occur.

From my own experience, the changes have been mind-boggling. As I look back, I cannot perceive how I could have anticipated or even dreamed of the changes that have occurred. As I look forward, there's still no way to anticipate what will occur.

Definitions

In order to establish a common ground to discuss the subject matter of Coping with Change, I've called upon an old friend, The Dictionary, to make sure we're all working from the same hymn book. My source is "The Random House Dictionary of the English Language"--Copyright 1966. I hope the definitions haven't changed in the last 21 years. Actually, I had a good time playing with the words, one leading to another, as I developed this paper.

COPE: To struggle with or contend, especially on fairly even terms or with some degree of success

CHANGE: To make the form, nature, content, etc. of (something) different from what it is or from what it would be if left alone.

So if I were to use these definitions I could paraphrase the subject of the paper to say:

Coping with Change: To struggle or contend on fairly even terms or with some degree of success with something different from what it was or would be if it had been left alone.

That definition opened up a few questions about definitions. Why" struggle" or "contend" and what's meant by success? I looked up those words and that led to more questions, not answers,

STRUGGLE: to contend with an adversary or opposing force

CONTEND: to struggle in opposition

SUCCESS: 1. The favorable or prosperous termination of attempts or endeavors 2. The attainment of wealth, position, honors, or the like

So I'll redefine the title Coping with Change: To struggle with an opposing force that is different from what it was or what it would be like if it had been left alone, and, upon favorable or prosperous termination of my struggle I will attain wealth, position, honors, or the like.

In summary, if our endeavors can take advantage of the forces of change, we can become Rich and Famous. Simple, isn't it? Well, there are some caveats having to do with the rate and magnitude of change and being able to recognize what's really happening.

Examples of Change

Before going much further, let me give some examples of changes that I've seen. I won't get into "should have" and "could have" if I had only recognized what was happening because that would make me cry alot. Well, one example will give you an idea of why one could become emotionally upset. When I first moved to Los Angeles, I could have purchased land in the far out suburb of Anaheim for $500.00 an acre. Some fellow by the name of Disney, or something like that, bought it to put up some amusement facility. I could have--should have!--oh well. Hindsight vision is 20-20.

As you read the rest of this paper you might keep in the back of your mind the following questions and relate them to the changes that you are going through.

* Are you willing to contend with change?

* What are the risks associated with change?

* What barriers are in the way of your acceptance of change?

* What advantages do you get from accepting change?

* What advantages do you get from not accepting change?

So that you have some perception of where my ideas come from, let me give you some background.

I grew up with such things as an ice truck and an iceman that delivered ice so we could put it in the icebox to keep food cold. Milk was delivered by the milkman with a horse-drawn wagon. The horse knew the milk route and would walk to the next house while the milkman put our milk in the milk chute. There was no such thing as homogenized milk. "The cream always rose to the top" of our milk; we had to shake it in order to get the right consistency. There was no stereo. In fact, we had to

wind the Victrola to play records.

Then came the push-button radio. What a great advance! It was no longer necessary to move the dial. We listened to radio programs with such characters as Jack Armstrong, the All-American Boy sponsored by Wheaties, the Breakfast of Champions. (I ate them everyday, and I was a champion.) There was Little Orphan Annie, the Shadow, Jack Benny, Burns and Allen,--all playing to your imagination of what was going on.

Then came TV and away went the imagination. It was no longer necessary to conjure visions of what heroes, heroines, and villains looked like. You don't have to exercise your mind to describe the jungle or Grand Central Station or stormy weather. It's all there on the screen.

Movies were ten (10) cents, ice cream three (3) cents for one dip and a nickel for two (2). Ice cream sodas were ten (10) cents.

Well, things have changed from those days. Progress has been made. We have to struggle to adapt to and keep up with the accelerating rate of change. This struggle creates stresses for all of us.

More Examples of Change

There is also the change in the work environment. In December of last year I celebrated my 30th anniversary of being in data processing. I started with IBM as an operations trainee. In my wildest dreams I could not have imagined the changes that have taken place since then. If I were to try to imagine what will happen in the next 30 years, or 20 or 10 or even next week, I would be hard pressed to come up with anything approaching reality. Most of you will participate in that change. The magnitude of which will be much greater than what I've witnessed.

I would never have believed that the soul of data processing, the punched card, would go the way of the dinosaur. Vacuum tubes, the life blood of data processing equipment, would also disappear. Chips would no longer be defined as the pieces of paper that result when a hole is punched in the card. Security would change from being a screen surrounding the material being processed with a sign saying "Don't look--This is Confidential" to the elaborate password and encryption procedures we have today. The technological changes in size and power of hardware defy comparison.

Nor could I have anticipated the Age of Specialization we now have. In the early days of data processing, there was total responsibility for a job. I recall that I designed the job with the user, wrote the specs, wrote the program, wired the panels, tested the program, ran the job, and delivered the reports. The only things I didn't do were keypunch the cards and decollate and burst the reports. If something went wrong, there was only one person to blame.

I must confess that I did make some mistakes, I won't elaborate on those, although they did provide an excellent learning experience. I would describe them in more detail, however, the shredding machine removed all of the evidence.

I will relate one experience in working with our data retrieval system. I was responsible for a pole inventory program for a large Telephone Utility and during the course of processing for them, I lost 2000 telephone poles, (consisting of one box of punched cards) out of their 500,000 pole inventory (250 boxes of cards). The retrieval system consisted of going to the storeroom and getting the cards--a little different from today's data retrieval. It's not too hard to see why the turn-around time for jobs was measured in days, not in minutes or seconds.

Most Significant Change

The most significant change that has occurred is the change in the relationship of the cost of people to the cost of machines. Again, relating to my own experience, my gross pay in 1957, working for IBM, was $6,000.00 per year. (I supported a wife, two kids, and a house on $425 per month take home pay.) I felt guilty because I was being charged out as a contract programmer at $10.00 per hour. The printer, which could print at 150 lines per minute, was being charged at $20.00 per hour. That low personnel cost versus high machine cost made it incumbent upon us to use people to solve problems rather than using valuable computer power. "Throw people at the problem" was the philospophy of the time. Use your own people and develop everything in-house became the rule. We had the disease called NIH or Not Invented Here. How could anyone else have the talent, the knowledge, the understanding and the resources to develop and maintain our application programs? "If you need more resources, hire them!" became the battle cry of the data processing department. Do not buy anything outside!

Just a few facts as stated by recent government statistics establish why that philosophy has changed. In 1965, 80% of the data processing dollar was spent on hardware and 20% for people. By 1985, that ratio changed: 80% of the data processing dollar was spent for people and 20% for hardware. The total data processing dollars spent in 1985 was significantly higher than the expenditures of 1965.

New Business Generated

It is no longer sound economic policy to throw people at the problem. This change also proved to be a vaccine to the NIH problem and helped create a whole new business--Packaged Applications Software. It is not only acceptable to buy outside software, in many cases it is the preferred way to get applications installed at a reasonable price and within a reasonable time frame. Packaged software is not the solution to all data processing needs. There are still a significant number of custom requirements.

Packaged software with all of its advantages of cost and time savings presents the potential buyer with some significant decisions to consider. In most cases, a business or organization has some unique requirements which are not handled by the packages available. This brings up a major decision point. Should the business be changed to meet the requirements of the package, or should the package be changed to meet the unique needs of the organization? There are many advantages for either choice.

From the perspective of the supplier, it would be most advantageous if programs are not changed. Then the program can be maintained exactly as supplied--release by release. The problem occurs when there are dynamic and changing needs and it is desirable for the software to meet those needs quickly. The vendor provides a generalized system that attempts to meet the needs of most of the potential users in the marketplace. The package may or may not be able to meet all of the needs of the users. The vendor is faced with the classic inventory control problem posed in any basic business class. What investment is necessary to meet all of the needs of all of customers 100% of the time? What resources will be required and how big will a system have to be to attempt to meet everybody's requirements all of the time? Even if everyone's requirements could be defined and agreed to, the systems would have to be too large and too complex to be practical.

Vendors send out periodic new releases with new features requested by users as well as by their in-house staff. These releases are infrequent because of the logistics involved in getting new releases to customers, the testing requirements to avoid errors in the new product and the disruption caused at customer sites where new procedures may be required to handle new features.

From the users' perspective, there is the concern that the competitive edge or the unique personality of the business has to be changed to meet the requirements of the software. Users need the option to modify to meet the business needs and they need to do so in a timely manner while awaiting subsequent upgrades from the vendor. The big problem comes when a next release arrives. That's when the cry goes out: "Help, because a new release is coming".

A New Release! What to do? What to do?

There are a number of solutions to the new release problem. They range all the way from "don't change anything" to "let the computer handle it". The decision as to which solution to take depends upon the cost to the business; the cost of modifying your business procedures versus the cost of resources necessary to maintain the modified programs.

If programs are modified, then it is necessary to establish a procedure to control the changes that are made. This procedure can range from manual controls with manual checking to automated integration of local modifications into new releases.

People who change programs should document the changes and maintain an archive of changes for every program. This archive serves as the source of changes that may or may not be necessary in the new release.

In theory, this should work well if programmers always document everything they do and if they do not overlook changes when visually comparing program listings. Unfortunately, with the significant demands on data processing people one of the first casualties is documentation. An often heard quote is "Today I'll solve the problem and tomorrow I'll document what I did." I've been hearing it and saying it since I've been in data processing - I have a room full of tomorrows still waiting to get done. I'm convinced that documenting of changes is number one on the "Procrastination Hit List." From a pragmatic point of view, programmers do not document. A more reliable source of documentation is required.

Software Tools

There are software tools to assist programmers in the documentation process. These tools provide an automatic audit trail of all changes made to a program when used as part of the job stream for moving a program into production. Not only do they satisfy the requirement for documentation of local changes when a new release arrives, but they also satisfy the auditors' concerns about control over changes made to programs. Whenever a program moves to a production group, the current production version is compared to the new version from the test group using a source comparison utility, and a report is produced identifying the lines of code that have been changed. This report must be reviewed and approved by the person responsible for the system - someone other than the programmer who made the changes. Thus control is established in the handling of changes. These hard copy listings of changes supports the effort to integrate local changes into a new release from the vendor.

Another method to support the integration of local modifications into the purchased software is to use a source comparison utility to compare the original vendor release with the latest production version. The output report would list all changes made since the prior release. The same original release would be compared with the latest vendor release to identify all changes made by the vendor in the current release. The two sets of output reports would then be reviewed. The machine finds all of the changes. The local staff has the responsibility to select the appropriate ones to be applied to the new release.

In choosing the source comparison utility to handle this

audit trail function, there are some features that will make the tool significantly more effective. The two most obvious features are ease of use and reliability. Check for those capabilities by trying demonstration programs usually available from vendors. In addition, the source comparison tool should have an algorithm that does not depend upon sequence numbers so that the programs do not get "out of sync". It would also be useful if the utility can identify moved code as well as code added, deleted or changed. Another time-saving feature is the ability to handle 'wildcard' comparisons. That is, file sets can be compared as well as individual programs. One entire release can be compared with another and look at all programs changed. It is also possible to get a summary of which programs have been changed and which have not been changed. This makes it possible to scope out the effort and resources required to implement the new release.

An even more comprehensive method to handle local modifications allows the machine to perform the comparison and integration functions. The tool compares the original release with the current production version and with the vendor's new release. An output report identifies changes made locally and changes the vendor has made. A compilable source module is produced which contains both sets of changes. Then an editor provides a means to edit out those lines of code that reflect duplications of functions. The final results is a source program that contains all desirable features and is ready to compile.

The capabilities offered by these types of utilities simplify the task of integrating local modifications into new releases of purchased or distributed software. These tools allow the machine to perform the tedious manual effort of identifying and applying changes, thus saving valuable programmer time. By letting the machine take on these functions, programmers can spend their time doing what makes them more productive--programming.

The functions provided by these tools allows for the maintenance and control of local modifications. Thus, the purchaser of application software can take advantage of the resources and enhancements supplied by the vendor and still have the flexibility to make desirable custom modifications.

These software tools provide the necessary "help" when "the new release is coming."

I believe that we all recognize and accept that change is inevitable. If we can develop the procedures to control change, then we can focus on the benefits derived from change rather than focusing on the discomfort that might be encountered along the way. We should look at change as an exciting challenge to be taken in stride on the road to success. And, as defined earlier, when we achieve success, we will become Rich and Famous. Simple, isn't it?

Visual Literacy : The Language of Visual Data

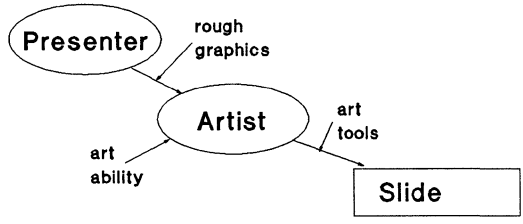
Joe Malin
Hewlett-Packard Company, Personal Software Division
3410 Central Expressway
Santa Clara, CA 95051

Computer software makes very sophisticated presentation graphics available to any computer user. However, the replacement of graphics artists by software means that knowledge of the visual language is no longer readily available. One does not need to be an artist to have enough knowledge to make effective presentation graphics. The important aspects of graphics for presentations can be summarized as principles of visual literacy, the most important of which are emphasis and integrity.

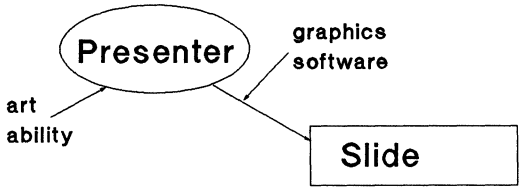
This paper describes how the principles of emphasis and integrity cover various graphics rules, and how these principles can be applied in presentation graphics design.

The Problem

Computerization has provided graphics to more people, but inevitably lessened the overall quality of the result. This is due to a change in the process of producing graphics, which used to be



but which now is more on the order of



This situation is not unique to graphics. The same problem has occurred in document processing. In that area, however, many fairly standard rules, coded into computer software, have begun to alleviate the problem. Visual communication has some standard rules, and these have for the most part been included in modern charting software. However, it is difficult or impossible to include the same rules in a general purpose graphics package. There are also some important principles that cannot easily be included in software. Some significant end user "smarts" are still necessary.

The Goal of Presentation Graphics

Fortunately this problem is easiest to solve for presentation graphics. By "presentation graphics" I mean graphic arts in any media used to support a presentation. I will focus on graphics printed on 35mm slides or overhead transparencies although the same principles apply to more informal printed graphics and even video displays.

These graphics have in common the focus of informing an audience, often with the avowed goal of persuading the audience to choose a idea or course of action from a presented set of options. A single presentation is usually made, consisting of a speech accompanied by displayed "slides" or pictures. The graphics definitely support the verbal presentation.

Every presenter wants to be effective. Therefore doing graphics "correctly" is more simply making graphics that effectively communicate simple ideas to an audience, and persuade them to choose an idea or action.

Visual Literacy

Visual Literacy

Visual literacy is the graphics equivalent of computer literacy. Someone who is visually literate is not necessarily an artist. He does have an appreciation of graphic arts principles which lead to effective graphics.

Regardless of the way it is taught, I believe that visual literacy should be concerned with both presentation and viewing skills. In this paper I emphasize presenting skills. However, both within and without graphics, the process of learning how to present may involve both being a speaker and a listener.

The core principles of visual literacy have been described in several places. McGregor and Nelson use a simple theory of shapes and colors to explain how to achieve visual impact. Other authors focus on the more standard graphics arts principles of simplicity, unity, balance, and emphasis. Edward Tufte in his provocative book on chart design uses a list of aphoristic rules to define a chart "aesthetic".

Emphasis and Integrity

Fortunately, all of these approaches reach similar conclusions. Even more fortunately, most presentation graphics software packages seem to include the "right" rules. In particular the rules of unity and balance are well-automated; most charting packages can automatically scale and group the data to achieve a satisfactory chart.

User training lacks an understanding of emphasis and integrity. The principle of emphasis realizes that presentation graphics have a particular message which must stand out in order to be effective. Integrity "balances" emphasis in realizing that the graphics message should not be misleading. The problem with integrity in presentation graphics is not necessarily that the presenter is trying to lie, but that because of ignorance he misleads or confuses his audience.

The Problem Areas

The need for user visual literacy is also not uniform throughout the production of presentation graphics. The following areas seem to need the most literacy to be used effectively: Color and color combinations, layout, selectable aspects of typography, chart type selection, and chart design.

Color and Color Combinations

Color is the most misused aspect of presentation graphics, and yet has a very powerful effect on the viewer's ability to perceive a slide. Color is pervasive in graphics; everything in a slide will have a color, even if it's black and white. It's an appealing thing to the graphics creator, and has the most emotional appeal and effect of any graphics element.

The common mistakes made with color and color combinations are the use of too many different colors, and the combination of extreme colors. Many different colors become confusing and actually decrease the viewer's ability to receive information. The human eye cannot focus simultaneously on both blue and red (a condition which is deliberately exploited in 3-D simulation). High-contrast colors (from the extremes of the visible spectrum) emphasize an object while low-contrast colors (adjacent on the spectrum) actually reduce emphasis. High-contrast colors are called "complementary" while low-contrast colors are called "adjacent".

Effective emphasis requires that a small number of colors be chosen and used throughout the presentation. This provides a regular pattern of background and emphasis. Many different colors generate "noise" over which only extreme emphasis can be seen. The use of too many different colors can also lead to accidental emphasis and therefore loss of integrity.

A simple and useful technique is to choose color pairs and color triplets. A color pair should be two complementary colors, and a color triplet two adjacent colors and one that is complementary to the other two. This method can be easily organized with the use of a color wheel of 6 colors (plus black and white). The wheel can be hand-drawn (or plotted as a pie chart) using 2 colors from each part of the visible spectrum (red-yellow, green-aqua, blue-purple) . These colors are ordered so that low-contrast colors are next to each other and high-contrast colors are across from each other (see Figure 1).

Individual colors have commonly accepted connotations as well. If chosen properly, individual colors provide emphasis on a particular point. Random color selection can harm the integrity of a presentation. A familiar example of

this is the unplanned use of red in presentations, since red has historically been used in business to indicate negative monetary values. Some standard color meanings are

```
RED    -- danger/excitement (also "negative")
YELLOW -- cheerful/sunlight
GREEN  -- quiet/nature      (also "positive")
BLUE   -- thoughtful/water
```

Typography

Typography is the "graphics" of printing words. The vast majority of typographic rules focus on achieving simplicity, unity, and balance. The overall goal of typography is familiarity, often stated as the idea that "a new font to be successful must not be recognized for its novelty". The typographic rules of letter and word placement are well-codified and should be found in a quality graphics software package. The variables in typography that will affect emphasis and integrity are choice of font, type face, font size, and text placement.

The meaning of font varies, but as used in this paper it refers to a family of stylistically similar alphabet letters. Within in a particular font there may be various type faces (such as italic). For each type face there may be several available sizes.

Although graphics software packages contain many different fonts, only one should be used in a presentation. Font changes produce a contrast which provides emphasis, but it is better to use a contrasting style in the same font. There is no agreement on a preference for serif (i.e. Roman) over sans serif fonts (i.e. Helvetica), although a familiar font may be easier to read . One advantage of sans serif is that thin serif lines can be lost in the scaling or reproduction of the graphics.

Modern (sans serif)

Classic (serif)

- Example fonts -

A "square" type face should be used with italics reserved to provide emphasis. Different fonts can be mixed into the presentation to provide visual interest. These should not be relied on to convey printed information. Overall, excessive variation in type face reduces the ability to establish emphasis.

No more than 3 font sizes should be used. The smallest size should be at least 1/8 ''. These limitations both promote simplicity and provide for full visibility in overhead slides. A consistent pattern of font size use is also needed. Titles should be consistently one size and text consistently a smaller size. This method makes it easy to provide emphasis by changing to the largest font size.

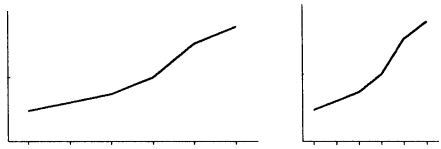
Lines of text on a slide should always be aligned parallel to the title, even in a data chart. In a text slide the lines of text are usually aligned along their left edges although the entire text box can be centered. No more that 5 words per line and 6 words per slide should be used. This reduces the amount of information on each slide, making it easy for the viewer to quickly scan the information. Too much text scattered about the slide forces the viewer to start reading, thus diverting his attention away from the presenter. Constant attention changes decrease the viewer's ability and desire to listen to the message.

Layout

The slide orientation should be horizontal with the base of the picture parallel to the long axis of the slide. The vertical orientation, while normal for close up reading, is more difficult to see over a distance. This may be because the eye has evolved to see variations from the horizon. A vertical slide makes the viewer "read" the slide rather than "view" it.

The horizontal orientation is also best for time-series charts which are the most common data charts used in graphics. The left-to- right progression of time is consistent with the "reading" movement and makes the data easier to understand. This also fits with the scientific standard for data charts which puts the independent variable (i.e. time) on the horizontal axis and the dependent variable (i.e. sales or profits) on the vertical axis.

Charts or graphics which have the most variability in the vertical direction (regardless of whether they are plotted horizontally or vertically) tend to exaggerate the variability of the data. Again, this is because of the mind's horizontal viewing orientation. Changes in position from bottom to top are more "interesting" and thus possess an inherent variability. A vertical orientation compresses this into a smaller left- to-right scanning space and thus exaggerates it:



- Data exaggeration -

The result is an unintentional emphasis which casts into doubt the integrity of the presentation.

The vertical orientation can be used in presentation graphics to make deliberate exaggerations. One should use this carefully to avoid an accidental "misstatement". One should also be aware of this fact when viewing presentations.

Chart Type Selection

The most familiar element of presentation graphics is the data chart. It is the oldest "presentation graphic" having made its appearance as a method of visual communication in the late 1700's. It is clearly beyond the scope of this discussion to examine data charts in detail; for this I would recommend a landmark book on the subject by Edward Tufte.

The visual literacy principles can be applied to chart type selection. For a particular chart, the factors of color, typography, and layout can be re-applied as they apply to data representation. In this respect a data chart can be viewed as a "slide within a slide".

Data charts should visually summarize a large body of data. This summarization has to be objective but also clear, and so it is particularly important that emphasis and integrity are balanced. To achieve any measure of either it is also important to choose a chart type that is common and familiar.

There are 6 major forms of data representation which in turn demand particular chart types. These are listed below, with examples in Figure 2:

Representation	Use	Chart types
Tables	Small sets of data or specific data	Data table
Maps	Relate data values to geographic areas	Zone map
Time-series charts	Relate data values to the passage of time	Column (bar) or line chart
Comparative charts	Show the relative values of two or more different data points	Column (bar) chart
Relational charts	Show the relationship between data points and often imply causality	XY Chart (scattergram)
Proportional charts	Show the relationship of different parts to the whole	Pie chart

Data tables although not really graphics can often replace more commonly used chart types. Comparative charts are often used to "prettify" a small set of numbers, when a data table might be easier to read. A comparative chart also loses the precise data values of a table. When these are put onto a graphic chart it becomes less clear and less readable, and the overall impact is lessened.

Data maps are a special form of chart that responds to a specific need. There is little need to differentiate their use from the other types of chart. Instead, one should note that restraint should be used in data maps. The representation of too many sets of data points in a data map can produce a visually pleasing but meaningless chart.

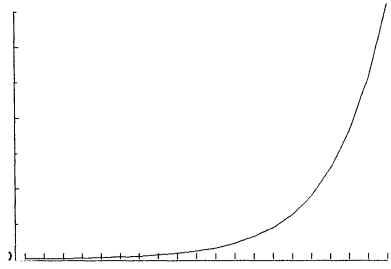
Time-series charts are a special but most highly used form of comparative chart. Tufte estimates that 75% of all chart graphics are time-series charts. There is a natural and understandable tendency to use the time-series chart since business data especially is collected in cyclical periods (fiscal years, months, quarters). The basic problem with time-series charts is that the passage of time is rarely a full explanation for data variability. The principle of integrity requires that more detailed investigations into the data be made along with a time-series chart.

Comparative charts display data points that are not differentiated by time. As comparative charts are usually represented by column/bar charts, they tend to emphasize stability or absolute value rather than change or relative value.

Relational graphs are the "purest" and most powerful chart type. The relationship between two sets of data expressed as written numbers

X	Y
0	1.00
0.5	1.41
1.0	2.00
1.5	2.83
2.00	4.00

is much more difficult to grasp than the equivalent graph on the x-y plane



However, this chart is the least familiar to a business audience. For the sake of integrity it should be used sparingly in a business presentation.

The pie chart is the prime example of a proportional chart. It is extremely popular and also extremely abused. In fact, some authors argue rather forcefully against its use, proposing other methods of representation. Pie charts can easily violate both emphasis and integrity. The common mistake of included too many slices lessens emphasis; the eye is unable to resolve them or receive the true nature of the data.

The violation of integrity occurs because of the eye's relative inability to perceive areas. The human vision system is twice as sensitive to changes in position as to changes in area. Furthermore, perception of change in position is significantly more linear than perception of change in area. As a result, the difference in two data values expressed as relative areas of objects appears to be smaller than the same difference expressed in relative position.

Data values are represented in a pie chart by the relative size of the central angle of the slices. There is no data on the visual perception of angle variation but the size of the angle affects the area of the slice, which means that the eye is also picking up area variation, which may be misleading. For similar reasons, two comparative pie charts should not be used unless they have exactly the same radius.

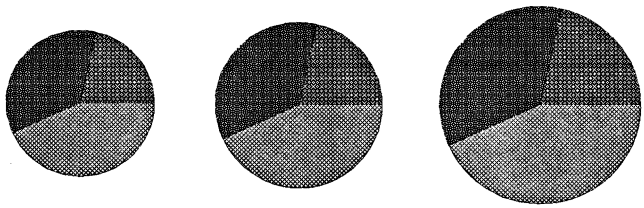


Chart B is 1.5 times larger than chart A by relative area. This rather large difference is not readily apparent. Also note that Chart C is 1.5 times larger in diameter (and thus appears significantly larger) but is nearly 3 times larger in area.

Chart design

The overall goal of a particular presentation chart is to present data in support of a particular idea or desired

course of action. The visual literacy principles are applied to the individual elements, which are usually under user control. The overall layout of the chart is usually under automatic control.

The overriding principle should be integrity. The basis of Tufte's entire aesthetics of chart design is chart "truth". I am in total agreement with this. A "true" chart emphasizes by its very integrity; a misleading chart by nature detracts from its message.

The individual chart elements that I will examine are data representations, axes and scales, and titles, labels, and annotations.

Data Representation

Many of the rules of data representation are built into charting software packages. Color, line, pattern use, and the number of variables per chart are not. Color should be used for focus, as described earlier. In a line, bar/column, or scattergram chart no more than three different sets of data should be used. In a two dataset chart complementary colors should be used, and in a three dataset chart two adjacent and one complementary should be used.

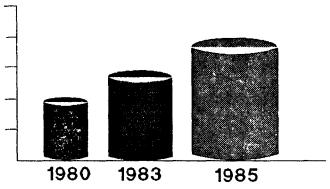
Area fill patterns in bar and column charts can also be used for emphasis. Extreme area fill patterns should be avoided since they cause "op art" patterns that are difficult to look at. Some patterns can also cause optical illusions which distort the data representation. Heavier or darker fill patterns tend to attract more focus. As in color, two similar patterns can be used with one contrasting pattern for focus.

Lines rather than columns or bars are used to provide emphasis, especially to accentuate change. The McGregor/Nelson paradigm proposes that diagonal lines imply drama and change, while straight lines and rectangles imply stability. This can be used for effect in chart design. For example, a line chart will focus on the variability of the data, while a bar chart will focus on stability.

Bars or columns should be rectangular and "flat" (two-dimensional). Design variations, such as 3D pictures that represent changes in a single data value, distort the data. This distortion is confusing and can detract from the

power of the presentation. Unfortunately these distortions can also be used to deliberately mislead the viewer by exaggerating variability in the data.

Consider the following chart which represents oil production volume for three different years:



As is common in such charts, all three dimensions are proportional to the volume amount, although this amount is actually a single dimension number. As a result the change between years is extremely overstated. The 1985 drum's volume in the chart is 5 1/2 times the volume of the 1980 drum although the diameter is only 1 2/3 larger and the height is only 2 times larger. The viewer has no clue as to which is the true relationship.

The data lines in a multiple line chart should be distinguished by colors in the same way as bars. If color is not available, line width is also a useful way of differentiating bars, with line patterns being the least preferable method. One problem with using line patterns is clarity; they do not scale well and are hard to differentiate.

According to the principle of focus, no more that 3 sets of data (i.e. 3 lines in a line chart) should appear in one chart. This allows the aforementioned triplet color scheme. If more than this number are needed, separate charts should be made. It is a common sense rule (often violated) that even slightly too much data in a chart can destroy the viewer's ability to understand any of it.

Axes

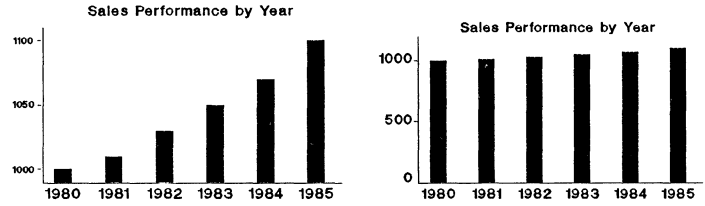
As in general slide layout, chart orientation should be horizontal. The axes provide lines of sight which serve as reference points for the eye to detect data variations, and should be horizontally aligned to improve viewing speed.

The vertical orientation is also less preferable because it tends to exaggerate the variability of the data. Again, according to the McGregor and Nelson paradigm diagonals connote change and instability. In a vertically oriented line chart the angle of each line is larger because of the compression of the horizontal axis. A similar effect occurs with the top line of column charts. This effect can be used for emphasis but may detract from integrity. The axis lines should be drawn with a thin line width which does not draw emphasis away from the data.

Scales

Scales give a context to the relative positions within a chart. It is crucially important that the scale of a chart be true. Scales should start at zero, except if there are negative data values, but then a zero line should be used. Using a non-zero starting point for a scale is misleading, even if the scale is clearly labeled.

As an example of a scale "lie", consider the following two charts which differ only in the starting point of the vertical scale:

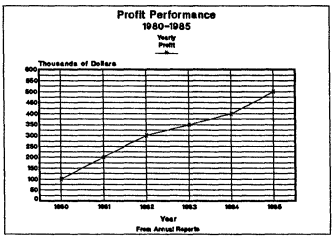


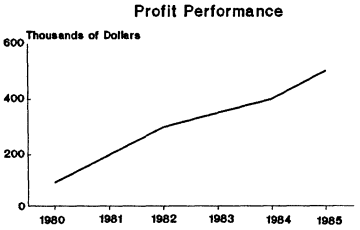
Tick marks (small thin lines perpendicular to the scale) should be used to uniformly divide the scale. The endpoints of the scale and the tick marks should be labeled (although Tufte favors reducing this so that only enough tick marks are labeled to make clear the magnitude of each division). Only one scale should be used for each axis. Multiple scale charts are marginally acceptable and tend to provide confusion rather than useful data.

There are rules for horizontal and vertical grid lines in charts but these are really "chartjunk" (Tufte's term) which detract from the clarity of the chart. As they are an option in most charting software packages, one has a choice. It is less easy to make mistakes in simple charts and easier to spot them. This is one example where the simplicity principle aids in achieving integrity as well.

Visual Literacy

Compare the emphasis on improved sales performance in the following two charts:





Titles, Labels, and Annotations

Titles, labels, and text annotations should follow the general rules of typography and layout that have already been described. A chart must have a title, and the axes and scales must be clearly labeled. The title should be at the top of the chart, and should simply and clearly identify the data being presented. Labels and annotations should be parallel to the title, in the same font but a smaller size. A neutral color (usually black) should be used.

Legends that identify the color or pattern of a particular dataset should be used with bar or column charts. Line charts and pie charts should be labeled directly with annotations.

General annotations should be avoided. If annotation of the precise data values seems absolutely necessary a data table should be used. This particularly applies to the practice of annotating the ends of bars or columns with the actual data value.

A Chart Design Paradigm

The chart design features I have described stress integrity. Both integrity and emphasis are best served by a simple chart. A complex and poorly designed chart at best diffuses its message and at worst can mislead. Tufte proposes a "data-ink ratio" which is the ratio of the amount of ink used to represent the data, to the total amount of ink in the chart.

Most charts have low data-ink ratio, some as low as 20% implying that 80% of the graphics does not convey useful information. The data-ink goal is to remove as much detail as possible while still retaining integrity. This is particularly easy to do in presentation graphics systems that either combine the charting and drawing functions or allow chart meta-files to be edited in a drawing package.

A Summary, and Some Implications

In summary, visual literacy applied to presentation graphics requires that the graphics be focused and truthful. Many of the techniques I have described can be summarized as instances of contrast. We are most easily able to see a message as a contrast against a steady background. It is therefore very important to know how to make contrasts and how to avoid them.

Visual literacy also has some implications for software designers and users. I hope that visually literate users will realize that the ever-increasing number of options in graphics software packages must be approached with intelligent caution. Designers should realize that options alone are not nearly sufficient to provide effective graphics.

In particular there needs to be more of an effort to provide a feature which would allow users to create a "palette" of options that apply to a set of slides. With this, users could set up a certain set of colors, fonts, font sizes, chart types, etc. from all the options available in the package.

Although graphics software packages may never replace graphics artists, a visually literate presenter using well-designed software can produce effective presentation graphics.

Figure 1. A typical color wheel

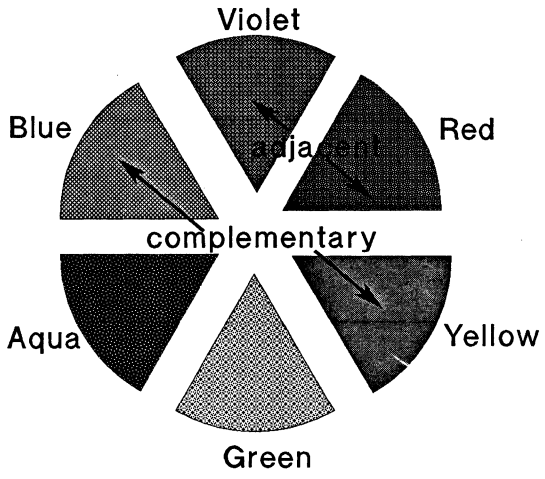
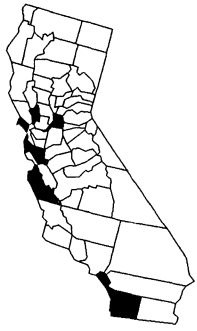


Figure 2 - Examples of Chart types

Data Table

Year	Sales	Profits
1980	100	50
1981	130	75
1982	150	80
1983	160	70
1984	200	110
1985	250	140

Map
(Zone data map)



Time-Series Line Chart

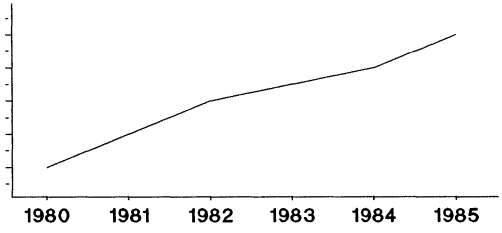
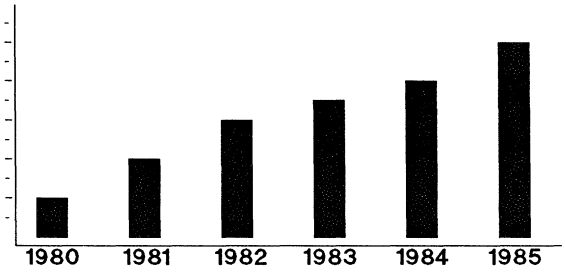
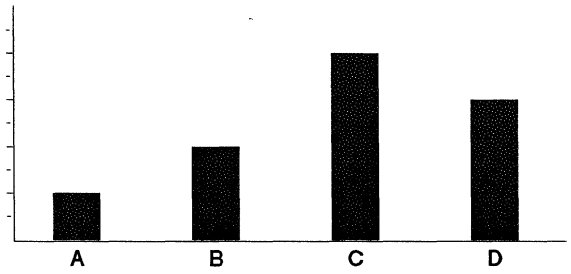


Figure 2 (continued)

Time-Series Column Chart



Comparative Column Chart



XY Relational Chart
(Scattergram)

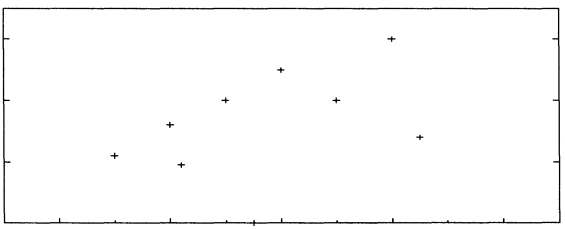
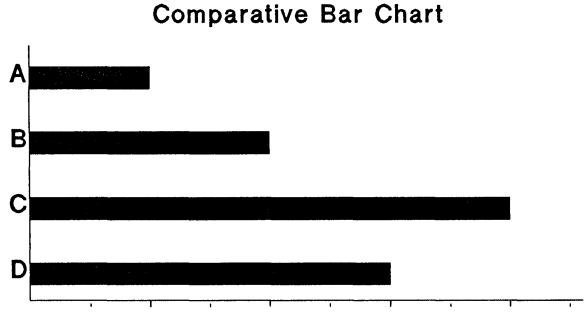
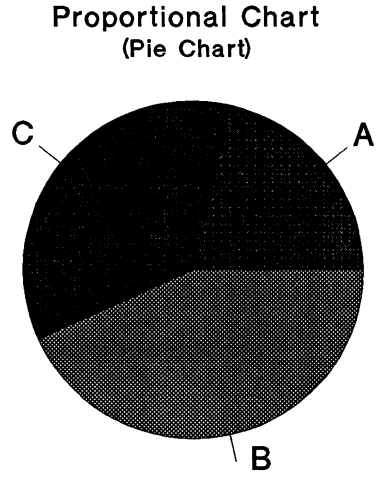


Figure 2 (continued)





References and Acknowledgements

The material in this paper is based on the following references:

Matkowski, Betty S., **Steps to Effective Business Graphics**, Hewlett-Packard Company, San Diego, CA, 1983

McGregor, S.L., and Nelson, M., "Use of Principles of Composition in Selecting Type Fonts and Graph Types in Business Graphics Applications", **SAS User's Group International (SUGI) Proceedings**, 1983

Meilach, Dona Z., **Dynamics of Presentation Graphics**, Dow Jones-Irwin Press, Homewood, Illinois, 1986

Reid, Brian K., **Scribe: A Document Specification Language and its Compiler**, Ph.D. Thesis, Carnegie-Mellon University, Dept. of Computer Science, October 1980

Simcox, W.A., and Ackerman, T.S., "Designing Effective Graphics: Human Information Processing and its Consequences for the SAS/ Graph(tm) Product", **SAS User's Group International (SUGI) Proceedings**, 1984

Smith, Wanda, "The Human Factors of Computer Color", to be published.

Tufte, Edward R., **The Visual Display of Quantitative Information**, Graphics Press, Cheshire, Connecticut, 1983

I would also like to thank Scott McGregor and Wanda Smith of Hewlett-Packard Company for their ideas and assistance.

This document was prepared with HP Graphics Gallery(tm).

Business Graphics : Micro vs HP3000

Jean Pierre Martin
Infocentre Corporation
3100 Cote Vertu
Suite 390
Saint-Laurent, Quebec
Canada H4R 2J8

INTRODUCTION

Nowadays everyone recognizes the value of graphics as an effective business communication tool. It is well known that graphic presentations communicate ideas and facts better than rows and tables of numbers. Corporate data represented by lines or bar charts can convey trends and exceptions more effectively and a lot faster. If everyone agrees with these facts, why then is it not more widely used in everyday communication?

THE HP ENVIRONMENT

If we look at the short history of business graphics in the HP world we find two categories of products : HP3000 products and micro computer graphic products.

At the HP3000 level, products evolved from DSG/3000 to HPEASYCHART, HPDRAW, and third party products like ARENS PRESENTATION GRAPHICS and more recently SPEEDWARE GRAPHICS.

In the micro world the HP community evolved from GRAPHIC/100 (Text chart, Bar chart, Line chart) to DIAGRAPH, PICTURE PERFECT to GRAPHIC GALLERY and SHOW PARTNER. There are now hundreds of business graphic products in the MS-DOS environment alone.

PRESENTATION GRAPHICS

This category of graphic is used to give management presentations, training sessions, product demonstrations and tutorials (see fig.1). A picture is worth a thousand words and this type of visual representation conveys and emphasizes the important facts better and faster than any text.

There is no question as to which class of computer is better suited for this type of graphics. Because graphic production is CPU intensive and multi-user facilities are not required, the micro computer is the better alternative.

DATA GRAPHICS

This is the class of graphic that conveys important facts about an organisation through the use of pies, bars, lines, scattered diagrams and even maps to represent the numerical values stored in the company data base(s). (fig.2)

The data gathering functions required to produce these charts often implies searching through large amounts of data stored in files or data base(s). This heavy I/O processing often requires the use and the power of a mini computer.

Let us address some of the problems encountered with "Data Graphics" like :

When done on a PC, data must be downloaded from IMAGE/KSAM/MPE files to the PC in a format compatible with graphic software. This task is often difficult, time consuming and non-SYSTEMATIC.

When done on an HP3000 a user needs to know how to navigate through various file structures to extract the desired information. This implies expensive serial reads of large data volumes. Time dependant trend analysis is not possible since yesterday's data is gone.

PRODUCTION GRAPHICS

This classification may sound curious to many. To my belief, this may well be due to the very recent introduction of the concept itself. My own definition of production graphics is one that can be produced as easily as a month end report.

What is needed to produce Production Graphics is a Complete Graphic System. The components of such a system are :

An automatic facility to gather and save summarized data.

Graphics usually represent summary data that are extracted from existing, printed reports. An automatic facility should exist to transfer these summary figures to the graphic package.

Another important aspect of having that "generic" graphic data base is maintaining the historical value of that data. By keeping January data (in summary format) in the Graphic Data Base it will be very easy, in twelve months to produce comparative result graphs (for January) again without scanning a huge amount of detail data (which is probably archived by then, anyway!)

One more important question is : why not extract that generic data as we produce the detail report(s)? Again this would prevent scanning the detail data file twice, once for the detail printed report and the second time for the summary data needed to produce the Graphics. Imagine having a command in your Production Report Writer, similar to a Print command, that says: 'Save the value of these Data Items (item list) in my Graphic System because my end users at some point will want to graph their values'. Inserting this command in regularly scheduled Production reports (Nightly, Weekly, Month-end Reports) eliminates the need to re-read detail data in order to generate graphic output.

End user Graphic Design capability.

Another important fact to be recognized is that even if a user does not know the format, type and coulor of the graphic needed, the generic nature of the data to be reported on is not in question. For example, "I want a graphic of Sales dollars and quantity, by product, by region, by salesmen."

A Complete Graphic Systen would have a facility to store that generic type of information in a Graphic DataBase, thus allowing the user to try many formats, types, colours, etc... without having to scan the detail data every time the graph is plotted. This, of course has the benefit of saving a a lot of valuable computer resources. All of this without losing any flexibility or imposing constraints on the design of the finished product.

Ability to catalogue Designed Graphs.

A facility to keep the various graphs permanently would be desirable. This would allow execution (on-line or scheduled) of any graphic design that is considered a valuable permanent ouput by the user.

Scheduling capability and Spoolable devices.

What is needed to produce these Production Graphics? First, you need a spoolable output device and must be able to schedule it in advance. The HP7550 (with its 150 pages tray and automatic feeder) and the Laserjets (no colors yet!) are spoolable graphic output devices that can be used for regular production.

With all of these components, a Complete Graphic System becomes a natural extension to the information reporting capabilities of your Fourth Generation environment.

THE MIS/DP DEPARTMENT VS THE END USER

Who is more knowledgeable to insure efficient extraction of data ? The end user can certainly define the data necessary to produce the graph, but a thorough knowledge of the Data Base structure is needed to define the best way to access the data. The Data Processing specialist (programmer, analyst or Data Base administrator) is the most qualified individual to extract the data from the Data Base(s).

The design of the graph, however, should be left to the end-user through a user-friendly interface. If access to the summary (generic) data is granted, we can afford to let the end-user try many alternative presentations of the data without using excessive computer resources. In this manner a presentation format can be chosen and the graph can be scheduled on a regular basis (daily, weekly, monthly).

For this type of graph, the data extraction function is, without any doubt an HP3000 task. Today, the design part of the graphs is still more easily handled at the HP3000 level. After all, transparent and efficient communications between the micro and the HP3000 are still in their infancy.

AND THE FUTURE

We are, without a doubt, seeing more and more graphic tasks handled by micro computers. With their CPU power they can usually perform feats that would not even be thinkable at the HP3000 level (The response time would be significantly degraded).

Today, the large data manipulations necessary to produce meaningful business graphics is still a task better handled by the I/O capability of the HP3000.

What will be seen in the future is a better hybrid solution that will more fully integrate the power of the micro and the HP3000 computer into one very friendly tool to produce Graphics as easily as we can produce reports with end user Report-Writers today.

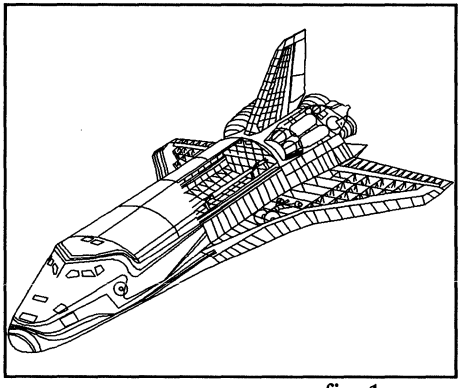


fig. 1

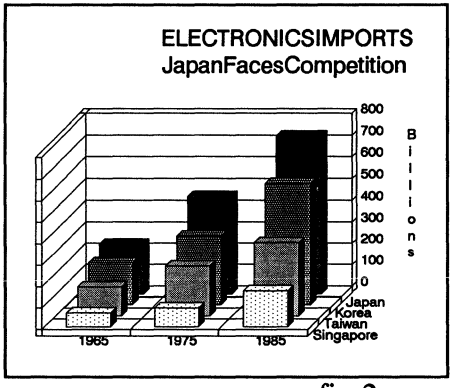
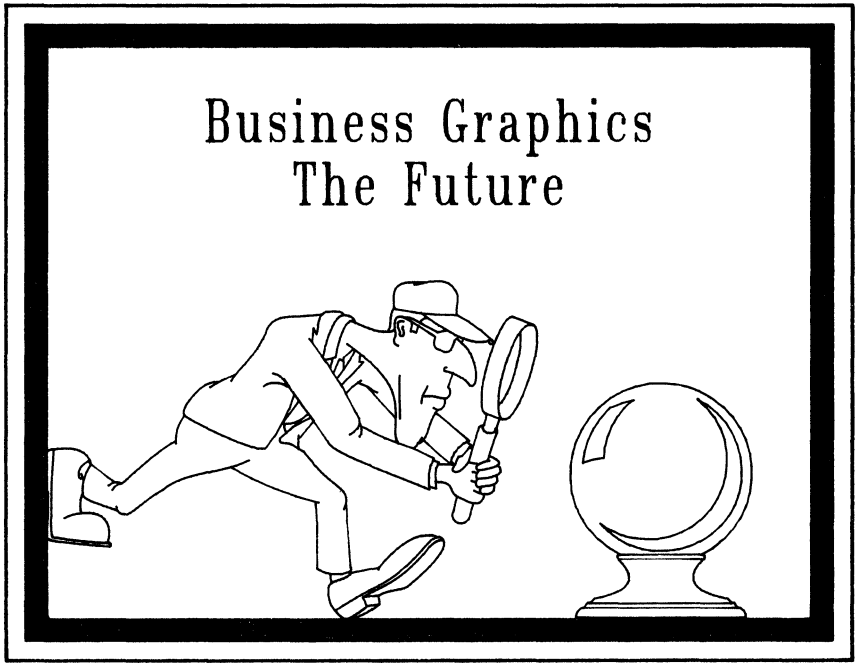

ELECTRONICSIMPORTS
JapanFacesCompetition
800
700
600
500
400
300
200
100
0
Billions
Japan
Korea
Taiwan
Singapore
1965
1975
1985

fig. 2


Business Graphics
The Future

Managing a Data Center Singlehandedly
(or rather, managing an HP3000 with a DP staff of one)

Karen Davis-Mackie
Cray Research, Inc.
5350 Manhattan Circle, Boulder CO 80303

When Hewlett Packard introduced the HP3000/Series 37 a new world of computers was available for smaller businesses. Finally, a fairly powerful multi-user computer with various business applications was available for around $20,000. And with the introduction of the HP Micro3000, more businesses can now have a computer servicing 10-20 users for about the same cost of a single-user Personal Computer 5 years ago! But to counter-balance the current declining cost of computing power, something must in turn begin to go up. This increase happens to be the cost of the employees working on the computer. Programmers, operators and other DP personnel don't come very cheaply to today's businesses. Therefore, small businesses and new system managers need to be educated as to the handling requirements of these small multi-user computers. What is required to properly run these small business computers? What does the person responsible for this computer need to know? And what should the company (and management) expect from this person in return?

As a person responsible for various data centers in the last 7 years (from a Series 30 to a 58, with a staff of 0-3) I have learned to effectively manage a data center singlehandedly by:

- Knowing my computer
- Knowing my users and
- Knowing my resources.

If the system manager is educated and informed in these three areas, they can remain effective (for the users), efficient (for the management) and in turn run a data center alone, or with very little help.

Know your computer

A system manager must understand both the hardware and software of the system they are supporting. A person strictly educated in only hardware or software will not be self-sufficient enough to properly service a small company. One of the best ways to tell if someone truly understands both aspects of running a computer (hardware & software) is if they have assisted with a new computer room installation. The process of selecting power and air conditioners, running

cables, configuring the computer and peripherals, getting the software installed and running, is all evidence of a person understanding the many angles of a computer's overall requirements.

In understanding the hardware a system manager must know the current system they are supporting (37,42,58) and what its benefits and limitations are. In other words, how much memory can I add or how many users can I add before this system becomes non-functional? Being educated about all models in the HP3000 line helps a system manager know whether to invest in the current model or upgrade to the next level when the current resources have run out. The system manager must also understand the environmental requirements of the computer. Keeping a constant environment with regard to power, humidity and room temperature, for any computerized equipment will extend the life of the machine and well as lower the frequency of repair. Also, when that new upgrade is finally approved and ordered, the system manager will know if the existing computer room will support the larger system. HP offers site preparation help in configuring an adequate computer room.

Specifically related to the current computer being used, the system manager needs to know:

- how to configure the computer (add that new disc drive!),
- how to properly back up the system,
- re-start or recover the system, and
- understand error messages and recovery methods.

A system manager coming from another vendor's shop will learn all of this very easily at the HP System Operator's class, or by going through the HP3000 Guide for the New System Operator manual.

As I mentioned earlier, the system manager must be knowledgeable about hardware. They must be able to talk intelligently with various vendors when purchasing computer room equipment, computers, PCs, terminals, modems and cables. To do this they must be knowledgeable about:

Electrical terms (Volts, 3-Phase, BTUs, etc.)

Computer room equipment (air conditioners, humidifiers, power conditioners, UPS systems).

Cabling specifications (RS-232 primarily for the HP3000, Coax, twisted pair)

Networks (from computer to computer if the company is large enough, or apread out enough, to warrant this).

Device Configurations of peripherals is mandatory if the users are not responsible for their own devices (terminals, printers, plotters). Helping users obtain the proper configuration for their terminal will be one of the first jobs a system manager has to accomplish with a new system installation!

One of the newest additions to the system manager responsibility list, and probably one of the most difficult to keep current on, is the area of networking and Data Communications. Understanding modems and the various aspects of data communications (Async, BiSync, 19.2, 9.6, Muxes, Line drivers) is very important to the data center of today.

Equally important with knowing about the hardware, is understanding software concepts in general, as well as the current software running on the computer. Understanding programming logic, database structures, flowcharting and file structures are all good concepts to know as a system manager. A person familiar with the type of software packages running on the computer (ie: General Ledger, Payables, Inventory, etc.) will also be far more beneficial to the company over someone who does not understand the existing software. If the company has chosen a system manager that basically understands the uses and concepts of the existing software (ie:understands financial statements, inventory systems, payables) the system manager then only needs to become educated about the specific software packages.

The system manager needs to find out where the existing software came from (was it written in house, purchased from a 3rd party) and who installed it. Also, who is responsible for supporting this software and what language is the source code written in? If source code is installed on the computer, can the system manager modify the code (time to order that compiler!)? How will upgrades or changes to this software be handled?

Specifically related to HP3000 software, the system manager must know MPE commands, know how to use the system utilities (where would we be without SPOOK?), and understand the HP accounting structure (accounts, groups, users). To be really advantageous to the company they must also understand system security, how to monitor system usage, and be able to document the system statistics (to prove the need to order that upgrade or new disc drive!). All of this comes with TIME and experience as a system manager (and attending the HP System Manager class helps too). Understanding the HP

specific products, such as IMAGE, QUERY, FCOPY, DS/3000, Inform, etc., are all added pluses if the new system manager is already familiar with them.

In my experience as a system manager, I have found the need for the use of flexible programming structures, and for occasionally having programming done by someone else. With only one person running the data center, programming is quite often contracted to people outside the company because I simply don't have the time to write and de-bug lines and lines of code. Therefore, I must be knowlegeable as to the programming specifics (I am essentially the analyst), but I don't spend the time writing the code. Make sure UDC's, LDev numbers, or Account security will not effect the way a program runs if any of them are modified. Sometimes moving a printer from Ldev #28 to Ldev #6 will make programs blow up all around you! And the users will be right behind looking for the system manager to fix the problem immediately!

Know your users

One of the most time consuming duties of a good system manager is the process of getting to know the users perspective of the computer. As a system manager, take the time to get to know your users. Get to know their limitations as to computer expertise, and become familiar with the products they are using on the computer you as a system manager are supporting. Make it a point to actually sit down and use the software you are supporting as a user. I have spent several days keying invoices, inventory activity, general ledger entries, and personnel letters among other things. I don't feel this has been wasted time, because suddenly I am able to understand the world as seen through my users' eyes. I am able to understand their language and am then able to talk to them using terms we both understand. If a system manager wants respect from the users, they must in turn show the users respect by taking the time to know more about them.

After learning the products your users are actively using, keep up to date on the products and always learn about any new software that is installed. Don't try to become the expert in every product, just be familiar with how the software works and where to find help if you need it.

As a system manager I have found that the more knowledgeable my users are, the better my job becomes. To help educate my users I try to keep them *informed* as to what is happening on the system.

Some ways of informing my users are:

monthly written reports (status reports) primarily for management information. These reports include system usage statistics, the current system hardware configuration, system issues, and my monthly activities.

Bi-monthly "user group" meetings (very helpful). This is usually a nice round table type of meeting where I can share information with all HP users at one time, and they can talk about any future plans or current problems.

Memos about computer room procedures. I use these to inform them about future scheduled down time, system back-up schedules, new procedures for using software or accessing the computer. This category also includes the ever necessary "cheat sheet" for those users that aren't quite independent yet.

Talking with the users and LISTENING to them as well. This has probably been the most beneficial method of giving information to (as well as receiving information from) the users. I have found out about more problems and actually solved them before they grew out of control by talking with my users than I can even mention. Keep in contact with your users and consequently they will feel more inclined to come to you with suggestions or problems.

Another way to inform or educate your users is by training them about the HP3000 or perhaps about basic PC information. I don't try to train my users on software specifics (I use classes, tutorials, or manuals for that) however giving them an overview about how the HP3000 works, or how their PC works will help them to not bother you when the only problem is that they have forgotten to put the correct diskette in, or their keyboard becomes disconnected. Cheat sheets are another way of training the less confident user to become more independent. Independence is the always the key quality for every user on a computer that I am responsible for!

When I have gotten to know my users and the systems they are using, and I feel they are comfortable with the equipment they are using I set myself up as a "help desk" for the users. In other words, I become the company PICS number for any questions they might have. I might in turn call the Office Automation SE if they have a question about HPDESK, or I might refer them to Corporate if they have a question about the Accounting System. What I am doing, is giving them a one-stop question answering service so they don't feel unsupported or alone. I am not trying to "know everything". I am only trying to know if my users are being serviced properly or if they need help.

Finally, keep up to date with your users' "wish lists". Occasionally I can help my users with something they thought wasn't available, or I might already have a product on the computer that will help them. This always makes them aware that I'm on their side trying to help, and when the computer goes down or in some other way inconveniences them, they don't seem to mind so much.

Also, don't try to overpower the users with your knowledge or expertise as a system manager. The old "keys to the kingdom" theory has never worked in any data center I've seen. Sharing your information with the users helps them understand and only makes your job easier, and at the same time you will look better in the users' eyes! The average users of today are becoming much smarter about computers and need to be treated with a greater level of respect.

Know your resources

As a DP department of ONE (yes, that's still one), the system manager must know where to find help in all areas. Knowing all possible resources is probably the one factor that has enabled me to maintain a data center singlehandedly with very few user complaints. Don't ever try to KNOW everything, simply KNOW who or where to call!

Some of the best resources I have found are:

Periodicals and Manuals Such as The Chronicle, INTERACT, the SuperGroup Assoc. magazine and various PC magazines. Manuals for all new hardware is kept in a specific bookcare for ready access. All software and system manuals are kept right by my desk.

Catalogs I like looking through the new HPDirect catalogs, INMAC catalogs, vendor mailings and re-marketed hardware advertisements for competitive pricing, new ideas, etc..

HP Products Such as the Telesup account, training classes (remember the System Manager class?), HP documentation (the Application notes and Response center ?'s and answers I receive are real informative)

HP People My best resources are usually my SE, CE and Sales Rep for help on a specific problem, or recommendation for an upcoming project. And don't forget to call PICS for system help!

Users Such as the Regional Users Groups, or the Int'l Users Group (love that Contributed Library!) are good resources. I have received help from system managers and users of all types of computers, not just the HP3000 folks. All users are potential resources.

Read Read everything you can. One of the best lunch time endeavors of mine is to keep current about products available for my computers by reading anything I might receive in the mail or from other users while I eat lunch. I read about other vendors products, programmers productivity products, new HP products and ways to help manage the data center. Keeping up to date about what is available out there is always a good recommendation to follow. That way you will always be able to tell your users about new products or methods to help out their department and make yours look good!

If at all possible, maintain the highest level of hardware and software support offered by HP. The account SE and 4 hour hardware support are necessary levels of support in a data center managed by one person. The users will be supported much quicker and your job will be much easier to accomplish if there are HP people that know you and your company, and can answer your questions quickly. Especially if you are expected to support quite a few HP products (like I am!).

Investing in system productivity tools is always good advice for a one-man(woman) shop. I recommend products such as Adager or DBGeneral for database maintenance, MPEX for an extended MPE operating system,and Report/3000, Inform, PowerHouse or other advanced tools for programming and reporting. Anything that will help make the system managers job more efficient and service the users in a timely fashion with a properly managed computer are worth looking in to.

Some helpful tools to keep on hand in the computer center, that will help save time and tears are:

screwdrivers - all sizes, especially the little ones for connecting cables

a break-out box - to know which signals are being sent by certain devices

spare cables - for those times when someone is testing new equipment, or has changed offices

gender changers - once again, for when people change equipment, or just want to try out a printer for a few days

RJ-11 wire/with clips - for those people that have modems

And finally, one of the best resources (finally one controlled by yourself) is using time management skills. Keeping track of projects, scheduling and attending meetings, and making sure the data center is running properly are all very time consuming duties for one person. Keeping the date centerwell organized will pay off ten times over.

I always keep the current copy of my hardware and software maintenance agreement handy which lists all of the device serial numbers. Beside each terminal or printer I write who is currently using that device. When a user calls me with a faulty terminal, I am able to call HP directly and know the model and serial number of their device. I also keep system logs beside the consoles, so I can write downthe down time and list the reason right on the log. That way I can know how much time the system was unavailable last month and why. Keep a current system configuration close by (use SYSINFO and SYSDUMP to produce this) and have a current list of resource phone numbers right by your phone. When I am out of town or home sick, my users can still get help by referring to this list. An even better suggestion is to publish this phone list and distribute it to a few "key" users. By keeping all of this information at your finger tips, you as the system manager will be able to complete more important projects and have time to learn about all of your users and the software on the computer.

Another reason to know other local HP users is to have a possible disaster recovery plan. Keep up to date about disaster recovery methods and try to incorporate some of them into your daily routine. I have taken backup tapes home daily, so as to have "off-site" storage in case of a fire (or whatever). I also try to educate my PC users to keep *frequent* backups of their hard discs in case of equipment failure or theft. Also, talk to other system managers about having their HP site be willing to serve as a backup machine for a few days in case of a disaster. And in return, offer your site to them. In a few days, when HP delivers your new equipment you can then restore your software and be up and running.

and finally **Know your limitations**

After learning about the computer, the users and your available resources, a system manager must begin to know their limitations. One person is exactly that, *one* person. They can not be expected to duplicate a DP shop with dozens of operators and programmers. Some limitations that are worth considering:

Keep training about software products the responsibility of the buying department. In other words, if accounting buys a

purchasing system, they are responsible to train the accounting users how to use the system. The system manager can sit it on this class (or demonstration) to learn about the product, but should not be required to actually train all of the users.

All Personal Computers connected to the HP3000 are exactly that, a *Personal* work station. The person using that work station is responsible for their own equipment. The system manager is responsible for the *system*, the person is responsible for the *personal* computer. As the DP department grows, this responsibility can be transferred over to a PC Specialist if the demand is large enough.

Finally, the management should not expect the system manager to KNOW everything right off the top of their head. It is an unfair expectation to demand one person to know all there is to know about computers (besides, management is not usually willing to pay the price for this sort of person!). An efficient system manager will not spend the time trying to know everything immediately. They will just spend time trying to know who to call or where to find the information for questions about things they don't know. As time goes on (experience, once again!) a system manager will be able to answer most questions right on the spot or be able to direct the user to the proper solution quickly.

An effective and efficient system manager will never try to "do it all". Trying to do *everything* will be like setting themselves up to fail. Running a data center singlehandedly means being the only person at that location that is responsible for the computer. That doesn't mean working *alone* or without other resources.

You, as a system manager, need to know your computer, know your users, know your resources and know when to say "I need help!". Maybe then, management will allow your DP department to become a prospering department of two!

The Development of the Red Roof Inns Communication Network

By: Ron Matheke and Pat Esposito

Red Roof Inns, Inc.
4355 Davidson Road
Hilliard, OH 43026

I. Introduction

The Red Roof Inns communication network has been a major project in the M.I.S. Department for the past three years. The demands and requirements for communications have been constantly increasing and more difficult to achieve on a cost effective basis. The network presently utilizes an application simulated on-line environment with an X.25 connection via CompuServe's terminal network.

This method of communication has been dictated by equipment and cost constraints. To develop our own true on-line network was too cost prohibited (excess of million-dollar capital investment and monthly charges > $100,000.)

The current method (attached Appendix A) has provided a simulated on-line network with a minimal capital investment at a substantially lower monthly cost. This has provided a true low cost communication network which is needed at Red Roof Inns.

II. Environment

Red Roof Inns

Red Roof Inns, Inc. is the largest privately owned and operated budget motel chain in the United States. There are presently 175 Red Roof Inns open or under construction throughout a thirty state area. The first Red Roof Inn opened in 1973 at Columbus, Ohio. All motels are company owned and operated and all are committed to provide a clean, comfortable, tastefully decorated room at a reasonable price. Red Roof Inns has one of the highest company-wide average occupancy rates in the lodging industry. The largest market segment is the business traveler, however, more family travelers and senior citizens have come to recognize the outstanding value Red Roof Inns offers.

Other Companies

Other major companies located within our Corporate Center are:

- J.R. Trueman and Associates, Inc.
  Construction Firm
  (Builders of Red Roof Inns)
- Red Roof Interstate Company
  Billboard Advertising
- Red Roof Development Company
  Site Selection Company
- TrueSports Company
  Indy Car Race Team
  (Winner of 1986 Indianapolis 500 and Cart Championship)

Red Roof Inns Communication Network

TrueSports Inc.
. (Mid-Ohio Race Track)

Computer Environment

The computer environment at Red Roof Inns can be broken up into two groups. The equipment at the Corporate Center includes the following:

HP 3000 Series 70 10 Meg Memory
(Reservations, 80 Users) and X.25 Communication (172 Users)

HP 3000 Series 70 8 Meg Memory
(Financial System) and other applications

HP 3000 Series 48 3 Meg Memory
(Programmer Development, Backup)

The equipment at each of our properties includes the following:

HP 1000 Micro 26 1 Meg Memory
(Front Desk System)

Communication Requirements

The basic communication needs revolves around our reservation system. In the hospitality industry this is a very critical application with requirements of speed and accuracy. At Red Roof Inns these things are extremely important and essential to maintaining our occupancy. This is because of our high occupancy (national average > 84%) and the need to make remaining rooms available to the property and at our Corporate Reservation Center at the same time. The other communication traffic involves our daily inn reporting, payroll and some electronic mail.

Communications History

We have gone through various communication techniques in the history of our reservation system. At various points in time a reevaluation was done to bring us to the next level. The following are some of the methods we have utilized:

1. Corporate Center called inn via telephone every time they made a reservation.
2. Inn called via teletype to pick up Corporate Center reservations.
3. Developed automated communication exchange via watts to match teletype usage frequency. (Minimal frequency.)
4. Developed first X.25 communication with CompuServe establishing virtual connection after inn and corporate computer's logged into their system. (Establishing virtual connection sometimes was lengthy and hard to manage.)

Red Roof Inns Communication Network

5. Developed current X.25 communication with CompuServe. The virtual connection established by Host logging into CompuServe, requesting a local modem for inn, dial and establish connection.

Application

The following concepts were key in the design of our communication programs due to hardware, software and reliability of the network utilized.

1. Mod 10 CRC incorporation to insure accuracy of data.
2. Packeting sequencing to insure accuracy of packet sequence.
3. Data compression to reduce number of characters transmitted.
4. Event monitor to reduce and schedule applications for load balancing.
5. Database engine to reduce number of image users. (Note: We did not feel Image could handle 250 users on one data base.)
6. Utilizations of son processes to reduce number of MPE users. (Note: We did not feel MPE could handle an additional 170 sessions.)
7. Utilization of message files to handle communications between processes.
8. Utilization of various networks (X.25 network, X.25 multiplexor padbox, modem port).

The following objectives were accomplished utilizing four batch jobs and one event monitor batch job. In the four batch jobs a database engine and inn level programs were activated to establish communications with all 170 inns. Then the inn level programs established its virtual connection to the inn and activates appropriate applications based on event flags in the application record of the communication database. (Appendix B.) These events are triggered by the event monitor programs on an established frequency.

The following are brief descriptions of all major programs utilized to accomplish communications. (Appendix C.)

PCOM0100 - Father process started by the communication job. Used to activate and monitor inn control programs (PCOM0200).

PCOM0200 - Inn control program activated by father process (PCOM0100). Used to logon, communicate to virtual connection (HP-1000), activate and control various application programs.

PCOM1000 - Data base engine program activated by father process (PCOM0100) to handle all application data activity to data bases for its specified job.

PCOM0500 - Event monitor program which scans reservation activity and updates the appropriate application record on the communication data base.

The following programs are activated by the inn control program (PCOM0200).

PCOM0300 - Reservations and availability application program.

PCOM0320 - Daily inn reporting application program.

PCOM0330 - Spooler process application program which currently receives payroll and transmits electronic mail applications.

PCOM0340 - Time card (payroll) send application program.

Problems

This is not to say that we did not have our share of problems. The following are a few of the problems that we have encountered.

According to HP documentation, the only way that we could FOPEN INP ports was to have a 2334 multiplexer at each remote site. We decided this would be too expensive, so instead of placing a 2334 at each inn we just configured NETCONF like it had a 2334 at each inn. By doing this we ran into several other problems. The worst one involved the FOPEN. When doing an FOPEN, a call is placed into the network. After the network accepts the call, the HP3000 sends a packet that contains terminal and printer resets and form feeds. The HP3000 does this to clear the device on the 2334 in the network. Fortunately CompuServe was able to modify their network software to ignore this packet.

Once we got by that problem we started to encounter many problems during our logon procedure. Since CompuServe is a terminal network we had to code the entire logon sequence. While doing this we found that CompuServe only passes 7-bit, even parity data during the logon. We were not able to read any of this data. Consequently we had to logon blind! In our logon program we code like we are reading data, but all reads fail. After we get logged in, (sometimes after several attempts), CompuServe goes

into what they call "image mode". In this state the network passes 8 bit no parity data. The only problem with this is that the HP3000 INP strips the eighth bit off of every byte of data. They are in image mode, and we are only able to read 7-bit data passed through the network.

For some reason, still unknown to us, the HP3000 will arbritarily send out an ASCII 'Read Pending' message. This usually is not a problem unless one of CompuServe's logon retries has been exceeded. If this happens CompuServe will clear the call, which means that we will have to attempt the logon again.

After we finally got past all of the logon problems, things really started to get interesting! While we were running small loads on the system we had hardly any problems at all, but when we increased the load we started having all kinds of trouble. For a period of about three weeks we were having five or six system hangs and failures a day! This episode began on September 28, 1986 and continued through October 16, 1986. The first few days we thought that the hangs were because of our software. We spent our time looking for possible causes. We could not find anything that would cause a major problem. After one week we became an HP hotsite. HP was on site every day for two weeks. Once HP got here several problems with their operating system and INP driver were discovered. The hang was found to be caused by a problem in the way the HP3000 was handling SIRS. We had a section of code which would attempt an FOPEN on a terminal multiple times. When the program would get into this tight loop the system would hang. This happened because the priority of the program would be raised into the A queue. Thus the program's priority was higher than MPE's priority. Obviously the system would hang. We had a relatively easy workaround until a patch was available. We just changed the code so that the program would no longer perform the tight loop.

The system failure problems were not so easy to solve. HP started out by swapping INP board, cables, etc. None of this helped. HP then installed a bug catcher on the INP's. This program finally discovered a bug in the DSMONX program. DSMONX appeared to have an invalid addressing routine. When the program would do heavy processing it would in essence, fill up its stack and address data outside its boundry. This would cause various system failures. HP sent us a patch for this problem and most of our failures were finally resolved.

Things ran rather smooth for the next couple of months. We continued to add inns to the network with relative ease. Then one day in February problems started again. Thankfully we were not getting many system failures. Instead of system failures we started to get CS I/O error messages. The jobs also started to log off with 'out of virtual memory' error messages. Everytime we

would get a CSIO error it would create an INP ram dump file. We were storing this file and shipping it to HP in California. This paid off because HP found another bug in the DSMONX program. They sent us a patch, it was installed and both the CSIO errors and virtual memory problems went away.

Things ran very smoothly again until the middle of April. We were starting to reach the limit on our virtual circuits configured in NETCONF. Our virtual circuits were configured at 50 and we had between 48 and 49 users on the INPS. Problems started again and this time things were even more interesting. We were not receiving any error messages on the console but the HP3000 was sending out a level 2 reset every couple of hours. We also started getting 'out of virtual memory' error messages again. Luckily our S.E. was able to solve this problem in a couple of days. As it turns out, there was another bug in the INP. Whenever we came to within one or two of our configured virutal circuit limit the INP would send out a level two reset. This would cause all the inns to disconnect. HP sent us a patch, we installed it and both the reset problem and virtual memory problem went away.

At the time of this writing we do not have any major outstanding problems. We still have system failures once or twice a week, but nothing like before. We are also communicating with every inn in the network for up to 16 hours a day. Things have been fairly quiet for the past month and one can only wonder what is going to happen next.

Future

The future demands for communication is increasing due to the cost evaluation of other alternatives and the fact that the connection is available. These are some of the applications we are working on and hope to provide in the future.

Electronic Funds Transfer of Credit Card Data. This will eliminate the current float when credit card slips are processed.

Full Two Way Electronic Mail Processing. This will provide messaging and sending of reports to and from the properties. (Provide timely and reduction of mail and phone costs.)

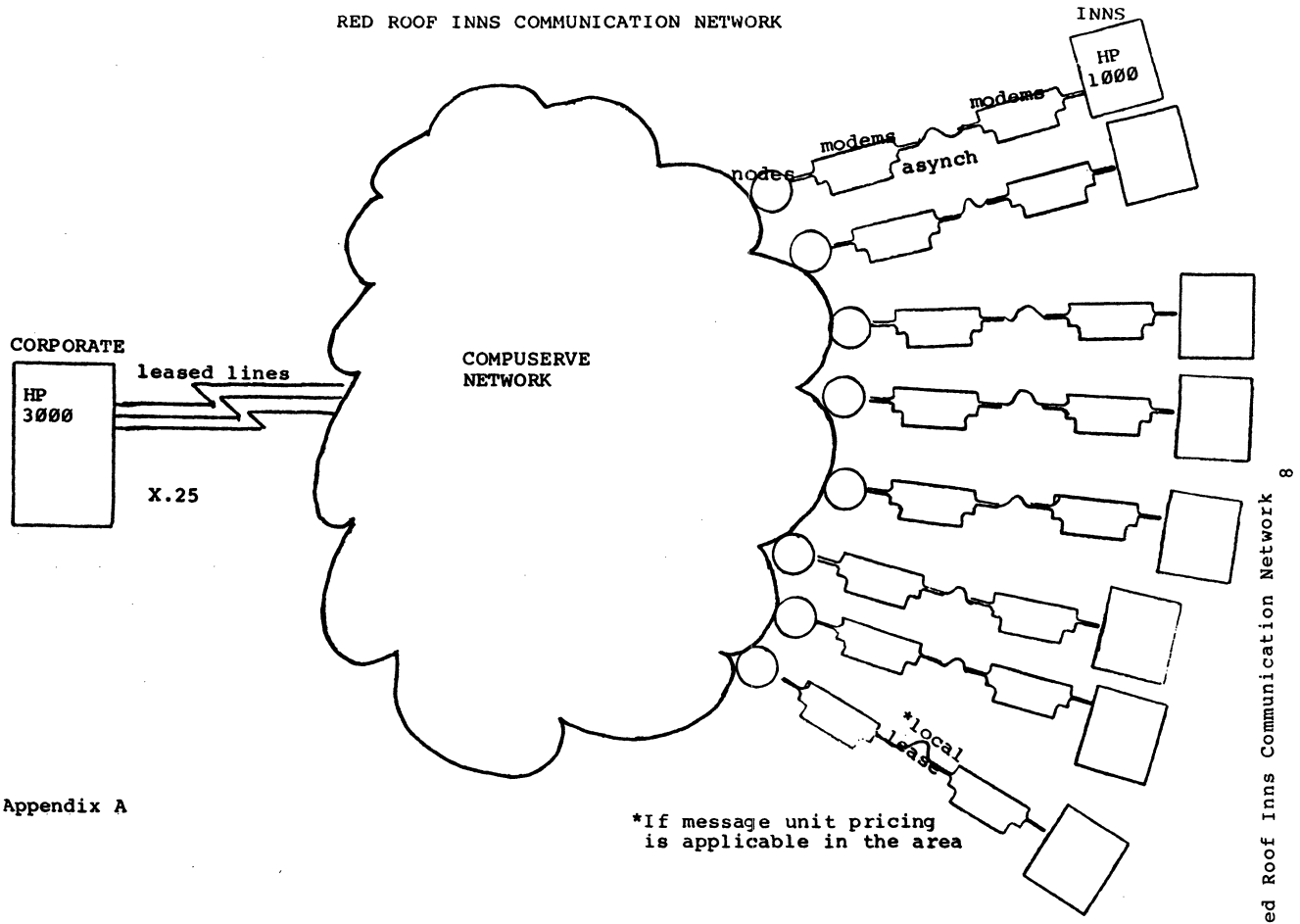
Front End Purchasing System for Ordering of Supplies. Provide up-to-date information of current parts, prices, and status of all orders.

In terms of actual networks, several new ideas are being reviewed. The utilization of satellite network is currently being reviewed. This potentially provides a substantial savings if we can provide satellite T.V. reception within the same receiver.

These changes and future potentials will provide actual cost savings to recover our investment and provide a higher level of service to our guests.

Conclusion

The development process has been long, rough, and many times frustrating. Even now, changes are being identified and make the system more efficient and to handle load balancing. However, we are beyond the stage where consultants and associates are second guessing each other if the concept will work. Bottom line, the network is now in a production basis and watching it work has made it all worthwhile.



Appendix A

Red Roof Inns Communication Network 8

APPENDIX B

COMMUNICATION DATA BASE

MASTER: INN-SESSION-CNTL

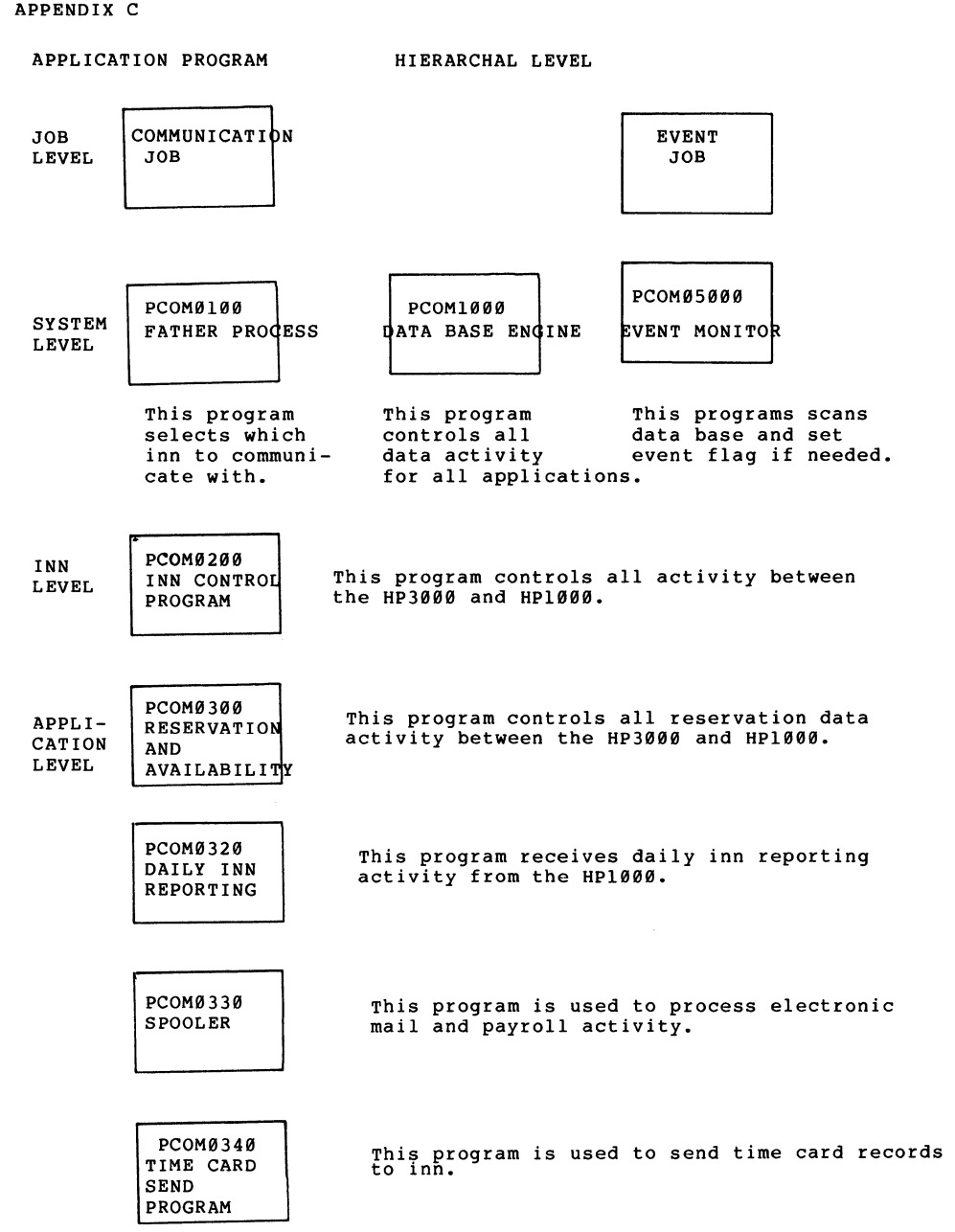
INN NUMBER	(KEY)
TIME OUT	DEFAULT TIME OUT PARAMETER FOR FILE READS
RETRY	NUMBER OF READ RETRIES AND DIAL RETRIES
SELECT	FLAG USED FOR INN SELECTION (Ø1-SELECT,ØØ-DO NOT SELECT)
POLL STATUS	SET IF INN IS CURRENTLY COMMUNICATING
CRC DATA SWITCH	PERFORMS DATA CRC IF SWITCH IS ON
PHONE NUMBER	PHONE NUMBERS USED TO DIAL INN
DIAL SEQ	COMPUSERVE LOGON SEQUENCE
INN FREEZE	FLAG USED IF INN AVAILABILITY IS FROZEN
START TIME	TIME THAT INN STARTED COMMUNICATING
START DATE	DATE THAT INN STARTED COMMUNICATING
MPE-SESSION	NOT USED
TIME ZONE	LOCAL TIME ZONE OF INN
COMPRESSION	PERFORM DATA COMPRESSION IF FLAG IS ON
INN NAME	NAME OF INN
FINISH DATE	DATE THAT INN FINISHED COMMUNICATING
FINISH TIME	TIME THAT INN FINISHED COMMUNICATING
WAIT TIME	TIME THAT INN IS TO WAIT BEFORE STARTING COMMUNICATING
DIAL TYPE	PULSE OR TONE
DIAL PATH	TYPE OF DIALING SEQUENCE TO USE
CRC HEADER	PERFORMS CRC CHECK ON HEADER PACKET
SHORT TERM	NOT USED
CITY	CITY MODEM IS LOCATED IN
MONITOR TIME	TIME THAT THE EVENT MONITOR FINISHED
MONITOR DATE	DATE THAT THE EVENT MONITOR FINISHED
ENGINE SWITCH	USED TO TELL PROGRAMS WHAT DATA BASE ENGINE TO USE

APPENDIX B

APPLICATION-SESSION-CONTROL (DETAIL)

INN-NUMBER	(KEY)
APPLICATION ID	APPLICATION NAME
PROGRAM NAME	APPLICATION PROGRAM NAME
DAY OF WEEK	DAY OF WEEK THAT APPLICATION IS TO RUN
SESSION FREQUENCY	HOURS OF DAY THAT APPLICATION IS TO RUN
GOOD APPLICATION DATE	DATE APPLICATION FINISHES CORRECTLY
GOOD APPLICATION TIME	TIME THAT APPLICATION FINISHED CORRECTLY
CONTROL DATA	DATA PASSED TO APPLICATION PROGRAM
SELECT SWITCH	SELECT IF TURNED ON
ATTEMPTED DATE	DATE APPLICATION IS STARTED
ATTEMPTED TIME	TIME APPLICATION IS STARTED
RESEND LIMIT	NUMBER OF TIMES TO RESEND DATA
MONITOR DATA	NOT USED (FUTURE)
MONITOR TIME	NOT USED (FUTURE)
EVENT FLAG	SET BY EVENT MONITOR
VERSION NUMBER	HP1000 SOFTWARE VERSION
SEQUENCE NUMBER	(SORT ITEM)

APPENDIX C


APPLICATION PROGRAM
HIERARCHAL LEVEL
JOB LEVEL
COMMUNICATION JOB
EVENT JOB
SYSTEM LEVEL
PCOM0100 FATHER PROCESS
PCOM1000 DATA BASE ENGINE
PCOM0500 EVENT MONITOR
This program selects which inn to communicate with.
This program controls all data activity for all applications.
This programs scans data base and set event flag if needed.
INN LEVEL
PCOM0200 INN CONTROL PROGRAM
This program controls all activity between the HP3000 and HP1000.
APPLICATION LEVEL
PCOM0300 RESERVATION AND AVAILABILITY
This program controls all reservation data activity between the HP3000 and HP1000.
PCOM0320 DAILY INN REPORTING
This program receives daily inn reporting activity from the HP1000.
PCOM0330 SPOOLER
This program is used to process electronic mail and payroll activity.
PCOM0340 TIME CARD SEND PROGRAM
This program is used to send time card records to inn.

AUTOMATED PROJECT MANAGEMENT
THE PLAN FOR SUCCESS IN SYSTEM PROJECTS

by

Robert R. Mattson
9545 Delphi Rd. S.W.
Olympia, Wa. 98502

INTRODUCTION

Project Management is like the weather and documentation....everyone talks about it but no one does anything about it! Maybe this is of an over statement but from my observations not much of one. This paper/talk is meant to shed some light on the whats, whys and hows of project management for "system projects". Additionally, I hope to provide reasons as to why managing such projects is very difficult without a good automated project management system.

SUCCESS IN PROJECT MANAGEMENT

I'd like to take a brief moment to describe what I believe is the general meaning of the word "success" when applied to system projects. My analysis indicates that over 80% of the measurement of success as judged by "top" management in a system project is based on whether it is "on-time" and "under budget". The remaining judgement factors are divided between having a minimally acceptable system, functional quality, lack of visible defects, etc.

It is not my current purpose to argue the validity of such a measurement methodology. In fact, I believe very strongly that such an approach is questionable from an organization/business standpoint. My purpose in addressing the definition of success is to emphasize why managing a system project for time and budget is so important. For a full discussion of the concept of "success" in system projects see a paper in the proceedings for the Interex Conference in Washington D.C. titled "Why Systems Projects Don't Quite Succeed".

THE PROJECT MANAGEMENT PROCESS

Well, if we know how success is measured, then what does project management have to do with achieving it? To answer that we have to define project management itself. Project management is the totality of "steps" undertaken by a project's leadership to assure that a project meets primary time, cost and quality goals. This is not a definition of GOOD project management. In fact, a simple definition of good project management may be impossible. I do believe that doing a good job of managing any project will however involve doing the following steps in an iterative manner.

Scope & Deliverable definition
Task planning
Task Dependency planning
Resource planning
Time planning
Cost planning
Resource "procurement"
Scheduling
Directing
Doing
Recording/Collecting
Reviewing/Re-estimating
Status Reporting
Redefining, re-planning, redoing, etc.

It is beyond the scope of this paper to discuss in depth everyone of the above facets of project management...these subjects cover entire books. I will however touch upon some of those aspects that make many of these project management tasks especially challenging in system projects. The value, to system project success, of understanding and applying good project management techniques must not be under-estimated.

AUTOMATED PROJECT MANAGEMENT SYSTEM

It is my belief that a good automated project management (APM) system is essential. A good APM system is essential due in part to the size and complexity of most system projects. I will when addressing each area of project management try to point out how a good APM system will help.

SCOPE and DELIVERABLES DEFINITION

Scope is the summary description of who, what, why, when, how and how much of a project. Deliverables are the more detailed definitions of what a project is to produce. The scope and deliverable definitions of system projects are characterized by one word "change". More than most other types, system projects change their scope and deliverables. In other words, the summary and the detailed descriptions of what we are going to create at the beginning differs significantly from what is finally produced. This is one of the reasons that system projects overrun their budgets of time and cost. In other words, if we end up doing more or re-doing too many tasks due to changes then it can't help but take more time and resources than we planned.

How can a good APM system help us to deal with this state of change. A good APM system will help us to clearly communicate the significant features of the project scope BEFORE a change. When a change is made it should be possible to more clearly show its relationship to the project as a whole. A good APM system can allow us to adjust our plans to what is "now" being asked. Further, by providing a consistent reporting mechanism we can communicate the impact in cost and time for proposed changes to our client

and/or management in a professional manner. The method of communicating the impacts of proposed changes will greatly influence how you as project manager or doer are perceived and judged by your client and/or management.

TASK PLANNING

Task planning is conceptually where "all" the tasks that will be done to produce the deliverables are specified. It is a critical step before doing task dependency planning, time planning, resource planning and cost planning. If we don't do a good job of specifying the tasks....then all the other critical types of planning outlined above cannot help but be WRONG! Yet, I will lay odds that this is seldom done adequately for even the most simple systems project (such as one program). If we don't do it well for a simple project how can we expect to do it well when the nature of the project is much more complex.

So, what stops us from doing a better job of this most critical step? Part of the reason is the number of tasks to be specified is very large. Trying to deal with this volume without automation is next to impossible. A good APM system will allow us to handle the volume of tasks required for most system projects. Another reason is that we do not appreciate the value of doing detailed task list planning. A third reason, is that we have no history that adequately gives us information about the steps required to accomplish a similar project/task in the past. As a consequence of this deficiency we are not able to "remember" what we or someone else encountered. We will discuss this in more depth in the section recording/collecting.

Another issue is -- how detailed should a task plan be? There is not an absolute optimum level. I can give some quidelines from my experience. First, when in doubt more detail is better than less. The only drawback to specifying tasks in "great detail" is when it takes "too much" time. In my experience "too much" is usually self evident. On the otherside, I've seen far more cases in which tasks were not specified in enough detail. As a consequence of this lack of detail task specification serious problems have arisen usually with underestimation of time and costs.

A good APM system greatly reduces the amount of time and cost spent doing detailed project/task definition. Without a good system the logistics make dealing with the detail almost impossible. A good APM system will allow us to define tasks in fine detail. This detailed definition of tasks will provide us with a much truer picture of what will be involved in doing the project. Good APM systems allow for the reuse of sub-plans and/or templates. This will encourage finer detail and more accuracy in task requirements. The ability to do top down type task planning is unbelievably powerful.

TASK DEPENDENCY PLANNING

Assuming we have done a good job specifying all significant tasks in "task planning" then we are ready to address "dependencies". Dependency planning is the step in which the relationships between tasks are specified and

reviewed. This is important because understanding the inter-relationships between tasks gives us the means to know what must come before what and what can't start before what is finished. Many times it is the lack of knowledge in this key area that leads to poor overall time and cost plans.

As in task planning, trying to do dependency planning without a good APM system is very difficult to impossible on any reasonably sized project. With a good APM system we will be able to specify inter task dependencies. We will be able to explore the significance of these to the project plan. We can better communicate to others through standard outputs the nature of the existing dependencies. The number of features which a good APM systems has to deal with task dependency planning is amazing.

RESOURCE PLANNING

Resources are the people and tools we use to get tasks done. We need to take each defined task and figure out how much of what resources for how long will be required to complete it. There are a number of concerns here. First, in systems work (such as programming), a person is not a person. By that I mean there can be a ten to one or greater factor between what two people with the same years of experience and same title can do in the same period of time. In fact, many times, the one person appears to be able to do something that the second "equally qualified and titled" person can't no matter how much time you give them. Secondly, without having a detailed definition of what the actual task to be done is, it is very difficult to estimate the amount of a required resource. Thirdly, if the first two concepts weren't challenge enough, we can throw in the fact that we usually have no accurate history of how long it took us to do a similar task in the past. Since this accurate history doesn't exist we can't apply anything but our "judgement" to our estimates.

So in what ways can a good APM system help? It can provide a history of the resource time and cost for similar tasks. It can give us quantifiable data on individual human differences in performing the same task. It can allow us the capability to compare what we planned for resources for a task against actuals used. With this knowledge we can use current variance data to modify the remainder of our plan. This can give early warning of problems to us and others. Further, a good APM system will allow intelligent allocation of resources. The conflicts such as over allocating a resource will be made visible. The nature of resources required will readily be seen. Further, the cost of each resource applied to a project will be evident.

TIME PLANNING

Time planning is when we figure out how long a task should take overall. It is the summation of resource times plus slack, waiting and contingency time. Time planning is in many ways an extention of resource planning. As we have seen, our ability to estimate how much of a resource for how long is hampered by many factors. We can add to this a few special twists provided by the fact that time is not a solidly defined attribute. To understand

this limitation we must think about what terms like a "week", "work-week", five days, seven days, 40 hours, 7 hours per day, "man-week", etc mean. My experience convinces me that they can have wildly different meanings depending on how, when and by whom they are being used. Yet, I see people using these all the time as if there was some absolute definition. The consequences of this confusion and fuzziness related to time is that we are never sure what it means when we see someones estimate of "how long" it will take to do some task.

Here is an area where a good APM system will shine. It knows about calendars, time and task/sub-task concepts. This makes determining the duration of a task or a task and all it's sub-tasks possible. Further, a good APM system will provide a standard for defining time. It will also provide a consistent means of recording and reporting time. These two features can by themselves greatly increase the accuracy and quality of time management in projects.

COST PLANNING

This area is one of the easier areas to deal with in project management. At least it is usually relatively easy to assign a cost per standard unit of measurement for a particular resource. Once this has been done then it is "merely" a matter of mulitplying our estimated resource units to be used by the rate to arrive at costs for a resource in a particular project plan. I will ask you to consider the problems with any project cost estimate based on incomplete lists of tasks and other inaccuracies we have reviewed.

A good APM system will make this task easy. A good system will have enough customizing capabilty to allow for producing fully loaded costs. It will make the calculating of planned project costs and actual costs to date a dream. It's ability to add remaining cost estimates will provide another essential feature to project management.

RESOURCE PROCUREMENT

A project must have the resources to implement the plan. To the extent these are "people" we are into a challenging area. If you are going to do it with experienced people you've worked with and trust...you are blessed. If fate says you must hire then I've got two suggestions. First, understand that hiring the right person is a very uncertain process. As a very experienced systems project manager said...if you bat fifty percent consider yourself lucky. I would add to this my feelings that the surest way to know whether a person will fill your need is to work with them. No question you can ask in an interview or of previous associates can take the place of a week of working with someone. Hence, my experiences tell me that even if you must pay them consulting wages....work with the person before you ever hire them. Then, if you have the luxury of not being under the gun to start, you can continue the search for the right person elsewhere. In the long run you'll be a million miles ahead to find the right person...not just a person! This is an area that is not helped much by any kind of APM system.

SCHEDULING

Scheduling is planning when tasks are going to happen in the short and long term. The product of scheduling is a schedule. A schedule gives an organized timetable. The goal in good project management is to make this schedule such that if it is followed then at least the project time goals will be met. Here is where things get fun. Without good scheduling I'm not sure how any project can get done in a highly productive manner. Yet, to do a good schedule requires that we understand and have done a good job of all the steps which came before. Additionally, scheduling is the shifting sands of project management! I have never seen a schedule for a significant system which did not develop major inaccuracies within a few days after it was made. And, if a schedule is not our best estimate of future realities then how much attention will anyone pay to it. In otherwords, how many of us keep old airline, train or bus schedules! So what does this mean? It means that we must be constantly revising our short term working schedules and as appropriate our long term schedules. Yet, if this is true, we would expect to see project managers and workers modifying their schedules on a daily basis.....how many are doing this? Very few! Why isn't this being done? Again, a major reason is that to do so with out a good APM system is almost impossible. Further, it is a chicken or egg situation. Without having the benefit of using a good schedule one does not appreciate its value. At the same time without a good APM system to help build and update schedules it is unlikely that one will get a chance to see their worth.

DIRECTING

In order to get people and resources to do a job there must be some direction given. The challenge is communicating directions in a clear manner. Good project management must let someone know what, when, why, and many times how something is be to done. As discussed above there are numerous challenges in specifying these attributes related to a task. I believe that many times it is this directing step that gets short changed. People need to better understand the limitations of human communication and ways to improve it. In project management the most significant factor is that far too many directions are verbal. Verbal communication is subject to large losses of information. Therefore, if all we ever do in a project is verbally tell someone about the who, what, why, when, how much and how then we are bound to have problems. I believe it is amazingly too common that persons assigned to a task have no clear idea of what subtasks, milestones, time, costs and other parameters are expected. Yet, many times we get much greater performance by just making a goal clearer for a person. People will tend to work much harder toward a well specified goal than a fuzzy one. Further, the more well defined a task/project is the less time it will tend to take to complete. This is because people more clearly know what they have to do and when they are done.

A good APM system will provide many standard "written" means to communicate about the project/task at hand. Without a good APM system the tendency is to revert to verbal with its severe problems.

DOING

The doing of a project is what we finally get around to after we've done all the other things discussed so far. If you're like most of us, you have a real tendency to start doing. I've already stated my bias for the fact that no matter how it may seem sometimes there is a better way..it's called planning and project management.

A good APM system should help one to do the planning and maintain it with the least amount of effort possible. There will be less tendency to start doing without planning when plans are relatively easy to do.

COLLECTING/RECORDING

What is collecting? This is what we should be doing in projects so that we know what tasks we really do and how long they take us. Most of the time what we do is try to fit our activities into whatever gross accounting categories or such have been established. We also are trying to play whatever game is currently on the board. These games include "I'll charge the next task for this tasks time so I won't be over budget". They also include "What I'm doing isn't on my task list so I'll put it in some task that is on my list". The games are endless. The result is that the data collected about projects is seldom of much value as a learning tool or reference database. We can't learn about what tasks we should have specified when planning because we gave no mechanism for recording new tasks that became evident. We have no means for knowing really how long a task took because we made the penalty greater for missing some plan than the reward for truthful time keeping. The net result is that this activity which is absolutely essential for improving our abilities to plan and manage projects is lost irrecovably. To increase the quality of project management means we have to come to grips with this tough issue.

A good APM system can help by providing the means to record what has occurred. It is outside the scope of a project management sytem to provide the policies and people management necessary to get good data on what is actually happening on a project. I believe a good strategy in this area has its basis in pushing the planning, recording, monitoring and reporting as far down in the project as possible. This means that in a sense everyone becomes a "project manager". When this becomes the case then there is a real tendency for better data to be put into the system. This is so because these lower level "project managers" see the benefits when they have to use it to do better planning.

REVIEWING/RE-ESTIMATION

Reviewing is the step which should be happening on no less than a daily basis by every participant in a project. Reviewing is tied to re-estimating. The step is to compare what we have recorded to date against what we planned to do. We also re-estimate what remains to be done and when it will be completed. In this simple concept lies one of the most effective techniques for getting projects done on time. This is a natural extention of the

concept discussed in the last section about making everyone on a project into "project managers". Thus, the key to this review/re-estimation process is to make it operate at the lowest level possible in a project.

In order to do this then we need to have good systems for planning, recording, comparing, reviewing and replanning on an ongoing basis. It should be fairly obvious that a good APM system will help in this task. Then, if the APM tool allows for a consolidation of these lower level estimates, we have the higher level view we need to see the big picture.

STATUS REPORTING

Status reporting is what we should be able to do on a daily basis. We should, if all the steps above were followed, be able to immediately inform our "clients" of the status of a project whenever they ask. The current norm is to give vague but comforting answers about "everything is ok" or such like. This may be followed up by periodic "glossy" presentations about how things are going. In most environments the latter is not worth the paper its written on....luckily most clients don't know the difference. What we really want to be doing is creating a project management environment where the tools, policies and methodologies for ongoing status reporting are in place. Then persons at all levels can be reporting in a consistent and efficient manner. Additionally, we must have a policy which places the responsibility for exception reporting on each person. By an exception, I mean any time there is "significant" differences between a current working plan and actual or new estimate. Then if an exception is found it is the responsibility of the person to alert the person he reports to in a specific manner.

REDEFING, RE-PLANNING, RE-DOING, ETC.

If anything is missing from most system project management it is the fact that the steps we have just described are not CONSTANTLY being redone. Project management is a iterative process. The state of the art is that we are lucky if we do it once. Yet, as should be clear from all I have discussed, doing a plan once for a system project is of extremely limited value and rather like a glimpse of a speeding train going out of sight into a tunnel. The key concept is that a good APM system will allow us to keep this iterative process. Without one we soon become overwhelmed by the process and tend to abandon it.

MICRO VS. MAINFRAME

There are automated tools available on both mainframes and micro computers. They both tend to have their strength and weaknesses. I believe the first step in making the choice of best tool is to understand the task to be done. Then we can evaluate the tools and choose the best one. That is a whole subject in itself.

SO WHERE DO WE STAND?

We have covered a lot of ground related to project management. The goal was to give an overview of what steps we should be doing in project management for our system projects. There are clearly many challenges. Yet, there are automated project management tools emerging which make it possible for us to conquer many of the challenges. Most of us will be asked to take on responsibility for system projects. Also, we are being expected to be able to predict how long a project will take and how much it will cost. To add further challenges, many of the projects today are of much greater complexity than in years past. The solution? I believe we must develop the knowledge and start using the tools that allow us to meet these new challenges. I believe strongly that the use of a good automated project management system will make the possibility of success in project management a reality. May your projects be managed!

THE TOUCH INTERFACE - THROW AWAY THE KEYBOARDS ?

by

Richard Corn and Robert Mattson
Washington Irrigation and Development Co.
1015 Big Hanaford Road
Centralia, WA. 98531

SYNOPSIS

This article explores the design and technical considerations in using the touchscreen interface in computer system applications using V-Plus and COBOL COBOL on the HP3000. The success of this interface and the relatively clean programmatic implementation makes a strong argument for its expanded use.

THE PROJECT AND ENVIRONMENT

The project we tackled was to develop a "Maintenance Management System" for our maintenance organization which would be useable by all potential users.

The system was to be implemented in a company that was definitely not high tech. WIDCO is a open pit coal mine employing 800 people. The maintenance departments for heavy equipment and coal processing account for 250 people. The maintenance function operates 24 hours a day seven days a week and is spread over 10 square miles. The average user is a mechanic who has never touched (no pun intended) a computer in his life.

THE CRUX OF THE PROBLEM

The desire on the part of management was to have a system which would be useable by ALL maintenance personnel without the need for intervening "computer sophisticated" maintenance planners. This posed several problems:

* No "standard" interface approaches we had used before would allow for implementing such a system in this type of user environment.

* Standard interface approaches would require extensive efforts to train and retrain users.

* System support staffing in non 8-5 hours would be unacceptable and too costly if standard interfaces were used.

* Low computer sophistication of the "average" user would make overcoming "computer phobia" a difficult task. Our experience lead us to believe standard interfaces would result in little or no use by the majority of our "users". This was not acceptable if the system's benefits were to be realized.

SOLUTION DESIGN CONCEPTS

Faced with such a project, user environment and interface problems we decided to explore "new" alternatives. The most promising of these appeared to be the utilization of HP's "touchscreen" interface. The more we explored this interface method the better it appeared.

We did face two problems in using of the touchscreen interface. First, what new design concepts and issues are raised by applying this interface. Also, How do "standard" design concepts interact with this interface? Secondly, what is required from a programming standpoint to implement this interface. Our research was unable to find anyone using this interface on an HP3000 using VPLUS and COBOL.

We were able to answer both the design and programming issues in a successful manner. We will address the design and programming issues in order. We hope that the insights gained from these areas will be of help to others.

DESIGN ISSUES

Our design approach to utilizing the touchscreen interface was to start with techniques that we had used before. We found that for the most part these techniques adapt easily to the touchscreen. The touchscreen does however add some positive and interesting aspects to these techniques.

SOFTKEYS - We are all familiar with softkeys. Our system utilizes these extensively by taking advantage of the ability of the softkey labels that appear on the screen to respond to touch also. What is more interesting to consider is that under our approach the "read it, touch it, select it" of softkeys now extends to the whole screen! This gives the system designer the possibility of "softkeys" or more appropriately "touchkeys" appearing all over the screen. Thus "touchkey" boxes of all types can be put on the screen with words and/or numbers in them for the "selection" by the user.

MENUS - One of the first uses we put the touch interface to was menus. We normally employ menus in all our systems to insulate the user from the details of the programs, jobs and operating system. By utilizing "touchkeys" on the screen with menu selections we built upon the common touch interface approach of softkeys and our system wide use of menus. This proved to be an extremely successful combination.

HELP - We employ online "manuals" and "help" screens for our touch based system. We felt this approach was necessary due to the fact that the users and work stations were so remote and temporally dispersed around the clock. Utilizing the help subsystem, as with menus, builds on the single interface approach of touching what you want to do on the screen. Because the interface is so simple, we have found that the size of user manuals can be reduced. A good portion of user manuals tend to deal directly or indirectly with the interface itself rather than the application functionality.

INQUIRY vs INPUT - We strove to develop a system which could utilize the touch screen interface as much as possible as long as it made sense. There is a challenging area in utilizing this interface related to information inquiry. If the user must do very much switching between the keyboard and the screen then this makes the use of the system more difficult. Further we felt the majority of our 250 users would be in an "inquiry and a produce standard outputs" mode. For these users we have built a system in which ALL functions can be accomplished without resorting to the keyboard. This is done through a number of "types" of touch screens. One of these types of screens include "touchlists". Touchlists display a list of all possible choiches in boxes on the screen rather than requiring the user to enter a selction value or key through the keyboard. Where selection was necessary but the number of possible choices too large to be displayed on one screen we developed numeric touch keypads. Once again because these screens all use the same "read it, touch it, select it" concept the user finds them easy to learn and utilize. One place our goal of keyboardless users was difficult was in dealing with user logons. The operating system does not know about touchscreens. We solved this problem by employing a terminal monitor program technique for many terminals rather than the traditional logon. If the user must do much alphanumeric or text input then the touch-screen is not practical. There does not appear to be a good alternative to the keyboard. What we wonder however, as we consider most computer application systems we are familiar with, is how often they make the user deal with so much more complexity (keyboards, input of codes and keys, etc) than is really necessary to accomplish the function.

TRAINING - We had identified this area as being one of the systems major obstacles to success. What we found was that the touchscreen interface when combined with other techniques, some of which are outlined above, makes training a factor of 10 times easier than more traditional systems might require. We have found that for the great majority of our users the touchscreen interface is so "intuitive" that all one has to do is show them how to read a "touchkey" and touch it to get what they want. From there on the user can apply this technique with literally no further training. For neophyte computer users and for occasional users this interface makes approaching and using the computer MUCH less fearful than more standard keyboard based interfaces.

EASE OF USE - The central design issue for to the touchscreen interface is ease of use. The bottom line is that it is easy for the user to learn and use. For many functions it is far superior to the standard keyboard interface. If the user base are doing inquiries and especially if they are only doing them occassionally then the touchscreen interface is the design of choice.

TOUCH SCREEN IMPLEMENTATION

The design requirements for our application called for COBOL programs driving VPLUS block mode screens with the touch screen interface. We were relieved of developing our own methods to accomplish this when HP added a touch screen interface to VPLUS in the G.01.01 release of MPE.

The method used by VPLUS to interface with the touch screen is really very simple. When the touch interface is enabled, VPLUS sends an escape sequence to the terminal to place it into ROW/COLUMN touch reporting mode. Then, when the terminal is touched, it returns the row and column of the touch to VPLUS. VPLUS translates this row/column location into the field number of the VPLUS form field that occupies the same area on the screen that was touched. This field number is returned to the user program in the LASTKEY variable of the COMAREA. The field number is returned as a negative number to distinguish it from other codes returned by VPLUS. The field number is the field's number as assigned by FORMSPEC at design time, not the field's screen order number. If the row/column sent by the terminal does not correspond to a VPLUS field, the code -999 is returned. While there are numerous ways to access the touch screen, this scheme has the least set-up and sensing overhead.

ENABLING THE TOUCH INTERFACE

To activate the VPLUS touch interface, bit 0 of the SHOWCONTROL word of the COMAREA is set on. At the next VSHOWFORM call the terminal is put into row/column touch reporting mode. The user's touch is processed and returned by VREADFIELDS. VPLUS checks the terminal type to verify that the terminal being used is a touch terminal. If the terminal is not touch sensitive, bit 0 is ignored. If bit 0 of SHOWCONTROL is set off, the VPLUS touch interface is 'deactivated'. Deactivated means that the terminal remains in the touch sense mode but VPLUS returns the -999 code for every touch. This can be very confusing for programmers and users alike. The workaround for this problem is to send the terminal an escape sequence to turn off touch sense mode.

VISUAL TOUCH FEEDBACK

When the VPLUS touch interface is active (bit 0 of SHOWCONTROL is on), the VSETERROR intrinsic will toggle the error enhancement of a field on and off as repeated calls to VSETERROR are made for that field. This allows the programmer to give the user visual feedback when a touch is sensed. It is important to give the user immediate feedback to a touch. Since the user has touched a field on the screen, enhancing that field is the natural response.

BITMAPCNV INTRINSIC

Included with the touch enhancements to VPLUS is a new intrinsic, BITMAPCNV. We used that intrinsic in our routines to control the touch interface. This intrinsic will convert a 16 bit word to a 16 byte array that contains ascii 1s and 0s to indicate the value of the bits in the source word. You can change the 1s and 0s to a new pattern and convert the array back to the word. This is useful to control the bits in the SHOWCONTROL word of the COMAREA. When converting the array to a destination word, the destination word must be set to zero before the call to BITMAPCNV (un-documented).

TOUCH CONTROL SUBROUTINES

To facilitate the use of the VPLUS touch interface we created subroutines to handle the turning on and off of the interface. One routine called TOUCHON simply uses BITMAPCNV to set the SHOWCONTROL word bit 0 to 1 and activate the touch interface. The other routine, TOUCHOFF, uses BITMAPCNV to set bit 0 of the SHOWCONTROL word to 0. It also sends the escape sequence (esc -z0N) that will reset the touch sense mode of the 150.

TOUCH SCREEN LAYOUT

Our design of touch screen layouts has evolved into two variations. One, the 'option select' type of screen, presents a series of boxes on the screen that the user touches to select options and control the operation of the program. Boxes are created by stacking VPLUS fields on subsequent lines and treating them as a unit when processing the touch. This type of screen is typically employed with inquiry or report initiation programs. This type of screen looks and operates like the screens used in the 150 PC's PAM and File Manager software. The other type of screen, the 'menu select', arrays rows of summary data on the screen where each row represents more detailed data that can be viewed on a lower level screen. The user can touch the data of interest to cause the program to display more detailed information for the summary touched. This type of screen is used for touch menus as well.

Option Select Screen Example:
This screen has two touch boxes, BOX1 and BOX2. The labels in the fields are supplied by the program. The fields have an initial enhancement of 'IH' and an error enhancement of 'I'. The user sees two light green boxes with labels and if touched, the boxes become bright green when enhanced by the program.

```
******** ******** ******** ******** ******** ******** ******** ********
  Touch the Option of your choice:

          BOX11.....                      BOX21.....
          BOX12.....                      BOX22.....
          BOX13.....                      BOX23.....
          BOX14.....                      BOX24.....


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

field: BOX11      number: 1
field: BOX21      number: 2
field: BOX12      number: 3
   and etc.
```

Menu Select Screen Example:
This screen has data fields arranged in rows. This allows several groups of logical information to be displayed. The user can touch any field in the group of fields to select more detailed data or some other function. In the example each row of fields DATE, ORD and DESC display different 'orders' on the screen. To view more detailed information about any one of the orders, the user touches any one of the fields on that order's row.

```
******** ******** ******** ******** ******** ******** ******** ********
  Order Inquriy - Touch the Order for more information

  Date      Order Num      Order Description

  DATE1...  ORD1....  DESC1.......................

  DATE2...  ORD2....  DESC2.......................

  DATE3...  ORD3....  DESC3.......................

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

field: DATE1          number: 1
field: ORD1           number: 2
field: DESC1          number: 3
field: DATE2          number: 4
field: ORD2           number: 5
  and etc.
```

TOUCH RESOLUTION

The touch hardware can resolve the touch of single line fields with fair reliability depending on where the fields are placed and how well the term terminal is adjusted. Some fields may be difficult to touch sense and adjustment of the screen can minimize this touch sensitivity problem. Fields on adjoining lines that are taken together to represent a single logical touch (boxes or multi-line field groups) where the resolution is to two lines or more work very well. In cases where we have only one line of data fields, we have created dummy fields on the second row for greater sensitivity.

TOUCH SENSITIVITY ADJUSTMENT

Adjustment of the terminal is critical to good touch operation. The procedure is to display a screen with many touch fields both across and down the screen and to adjust the vertical/horizontal position of the screen image on the terminal until you get reliable touch sense on all fields. We have seen some variation in how well different terminals respond to the same screen. It is also important to keep the screens clean and in the case of small screen 150s, to keep the holes in the screen frame for the touch hardware clean and free of obstruction.

APPLICATION CODING OF THE TOUCH SCREEN INTERFACE

The program code to handle the touch interface is straight forward. After the call to VREADFIELDS, if the VLASTKEY variable in the COMAREA is less than zero touch processing is performed. The code checks for the -999 return code and if found displays a message asking the user to touch more carefully. If other than -999, the code falls though a series of IF statements to determine which field (or field group) was touched and proceeds accordingly. When a touched field is detected, that field (or field group) is enhanced and a call to VSHOWFORM is made to make the field light up immediately so the user knows the computer has registered the touch.

Example code for the Option Select Screen Example:

```
        CALL "VREADFIELDS" USING COMAREA.

        IF COM-VLASTKEY < 0
          PERFORM PROCESS-TOUCH.
              .
              .
              .

    PROCESS-TOUCH.
          IF COM-VLASTKEY = -999
            (set up and call VERRMSG with a 'touch more carefully message')

          IF COM-VLASTKEY = -1 OR -3 OR -5 OR -7
            (set up and call VSETERROR to enhance fields BOX12, BOX12,
               BOX13 AND BOX14)
            CALL "VSHOWFORM" USING COMAREA
             PERFORM BOX-1-PROCEDURE.

          IF COM-VLASTKEY = -2 OR -4 OR -6 OR -8
            (set up and call VSETERROR to enhance fields BOX22, BOX22,
               BOX23 AND BOX24)
            CALL "VSHOWFORM" USING COMAREA
            PERFORM BOX-2-PROCEDURE.
```

TOUCH SCREEN COPYLIB MEMBERS

To get away from hard coding field numbers in our programs, we wrote a utility that will create a copylib member for a screen that consists of a single word with subsequent 88 level variables that are the field names and their numbers. We move VLASTKEY to the word and then test for touched fields by name. If the form changes, we can simply re-run the utility to create an updated copylib member and re-compile the program. With the copylib utility, we can group fields together to create logical 'rows' or 'boxes' with a single name. This simplifies programming of box selection and 'menu' data selection screens.

Copylib Member Example for the Option Select Screen Example:

```
        01  TOUCH-SCREENNAME                PIC S9(04) COMP.
            88  SCREENNAME-TOUCH-MISSED     VALUE -999.
            88  BOX11                       VALUE -1.
            88  BOX21                       VALUE -2.
            88  BOX12                       VALUE -3.
              .
              .
              .
            88  BOX-1                       VALUE -1 -3 -5 -6.
            88  BOX-2                       VALUE -2 -4 -6 -8.
```

This simplifies the example program code:

```
        CALL "VREADFIELDS" USING COMAREA.
        IF COM-LASTKEY < 0
          PERFORM PROCESS-TOUCH.
              .
              .
              .

  PROCESS-TOUCH.
        MOVE COM-LASTKEY TO TOUCH-SCREENNAME.

        IF SCREENNAME-TOUCH-MISSED
           (set up and call VERRMSG with a 'touch more carefully message')

        IF BOX-1
             (set up and call VSETERROR to enhance fields BOX12, BOX12,
                BOX13 AND BOX14)
           CALL "VSHOWFORM" USING COMAREA
           PERFORM BOX-1-PROCEDURE
              .
              .
              .
```

USE OF THE 'PROTOS' 4GL

It should be noted that our application was written using the PROTOS 4GL. Use of PROTOS did not significantly affect the way the touch interface is coded but did greatly simplify some tasks such as enhancing all of the fields on a logical row or box.

TOUCH SCREEN 'BACKUP'

On most of our touch screens we have included a function key back-up to the touch interface. This consists of using a function key to step from field to field (or field group) enhancing them as you go. This allows the user to step through the logical groups of field(s) until the desired item is highlighted. A second key is used to signify selection and the program pro-

ceeds as if the field(s) were touched. The touch interface must be enabled to make the enhancements work correctly. Visually, this appears to work as many micro software packages that use arrow keys to highlight items on a menu. In keeping with our desire to provide immediate feedback to the user, we may enhance the function key label boxes in inverse blinking video in response to touching the label or pressing the function key. This inverse blink continues until new data is displayed or the function requested by the key is completed. Again, the the coding of this function is made very easy by PROTOS.

SYSTEM OVERHEAD

We have found no significant overhead problems with the touch interface in terms of the application or our cpu. The touch interface is minimally affected by the load on the machine. This can appear as a momentary insensitivity to touch or by a jerky appearance in the enhancement of fields in response to a touch.

REAL LIFE EXAMPLE

The example given here is the inquiry for our Work Backlog Tracking Module. Repair work requests or 'backlogs' are entered into the system and can be reported in a number of ways. The first thing a user must do to report the work requests is to select a subset of requests to view. The user does this by touching the appropriate box on the Backlog subset selection screen (see screen 1). The box touched by the user goes into inverse video full bright to indicate that the touch was registered (screen 2). Next, the highest level of our Equipment Index is displayed (screen 3). The user touches the item of interest (in this example, Rolling Stock Production, screen 4) and the next lowest level is displayed. The user proceeds through lower and lower levels until the final selection of equipment Sub-Class CAT 785 has been made (screens 5 & 6). With the the work requests subsetted by equipment type, the user can now futher subset by work priority, type and several other criteria (screen 7). The user touches the options desired and then touches SEARCH BACKLOG to start the extraction process (screen 8). When the work request backlog entries that meet all of the selection criteria have been extracted, they are presented in a summary list form (screen 9). The user can page through the list as well as print it in several formats. If the user wishes to see more detail about a work request, he touches the request on the screen (screen 10) and the work request detail is displayed (screen 11). In this example, a complex inquiry has been made without a keyboard, without having to remember codes or commands and the interface is natural and easy to understand and operate.

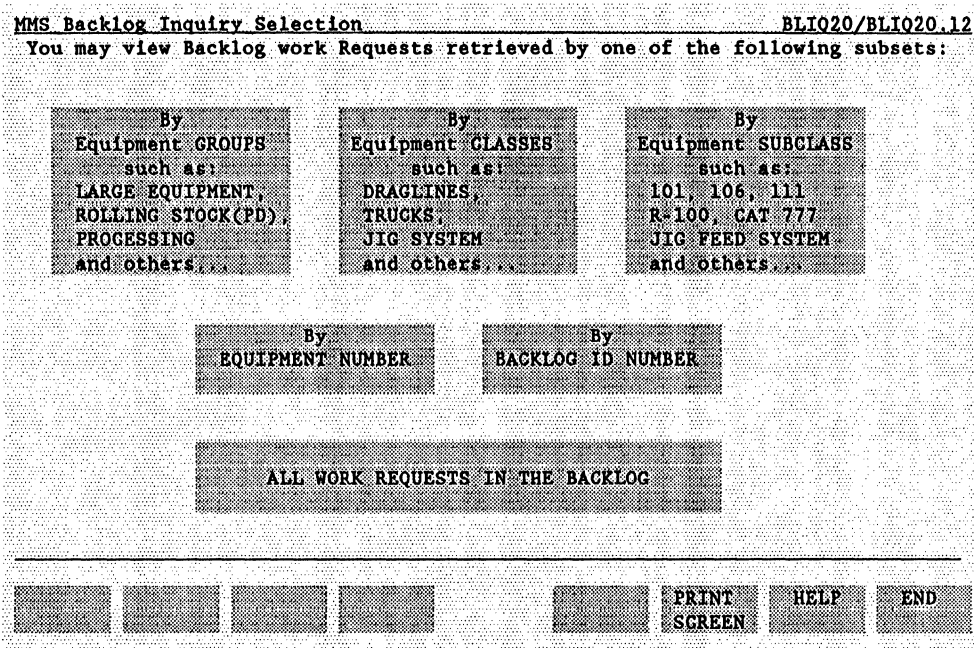
REFERENCES

The HP documentation of the VPLUS touch screen interface can be found in the Communicator, volume 2 issue 4. Our project was done with HP150 A, B and TOUCH SCREEN II models. In theory the HP2923 and 2927 terminal could be used with the touch interface but we have not tried these models.

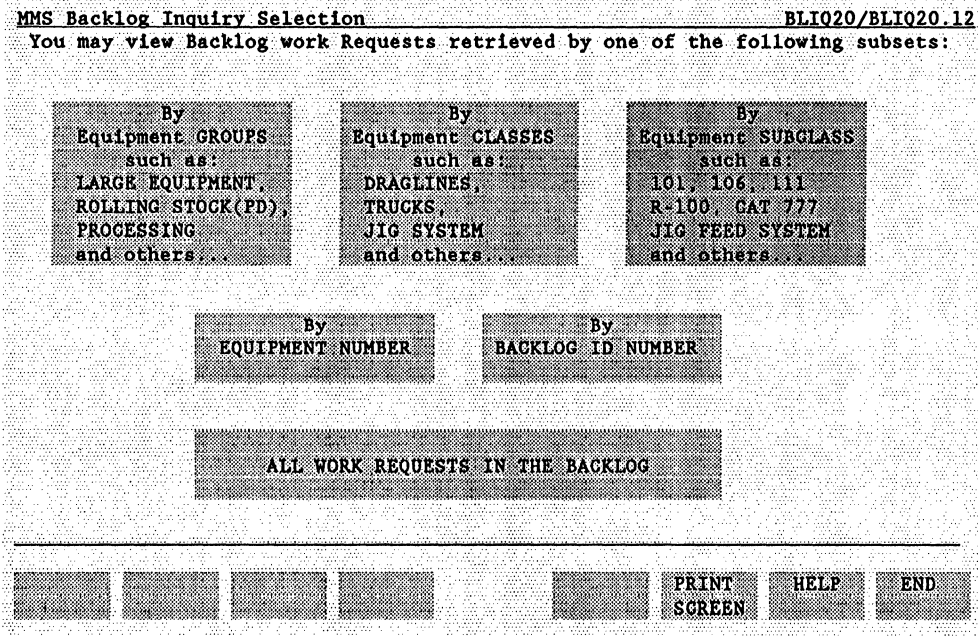
CONCLUSIONS

This large system is approximately half implemented. We have implemented four of the eight touchscreen based modules. The success of the system design concepts and the programming of the touchscreen interface has exceeded our expectations. We have been able to introduce this system into a difficult user environment with a minimal amount of training and "computer based" phobia type of resistance. The user response to the touchscreen interface has been highly favorable. A common response from these non high tech users is "it is so simple any one can use it". This rapid acceptance has greatly speeded up our implementation plans as the usual "break in" period almost disappears.

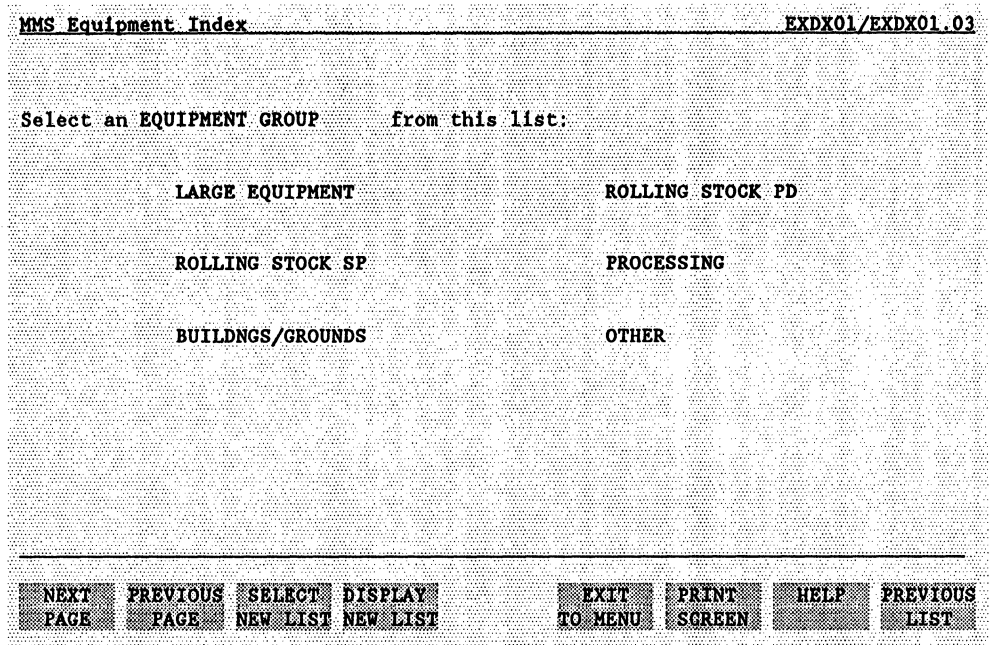
We believe that the touch interface has shown itself to be feasible from a technical point of view. More significantly is that it is desirable from a design strategy standpoint. Our experience with this methodology would indicate that its use could add significant benefits to many computer applications now in use or contemplated. As we expand the "users" of computer systems we will find significant benefits for these people in allowing them to touch their solutions. Do you have an application just waiting for the right touch?



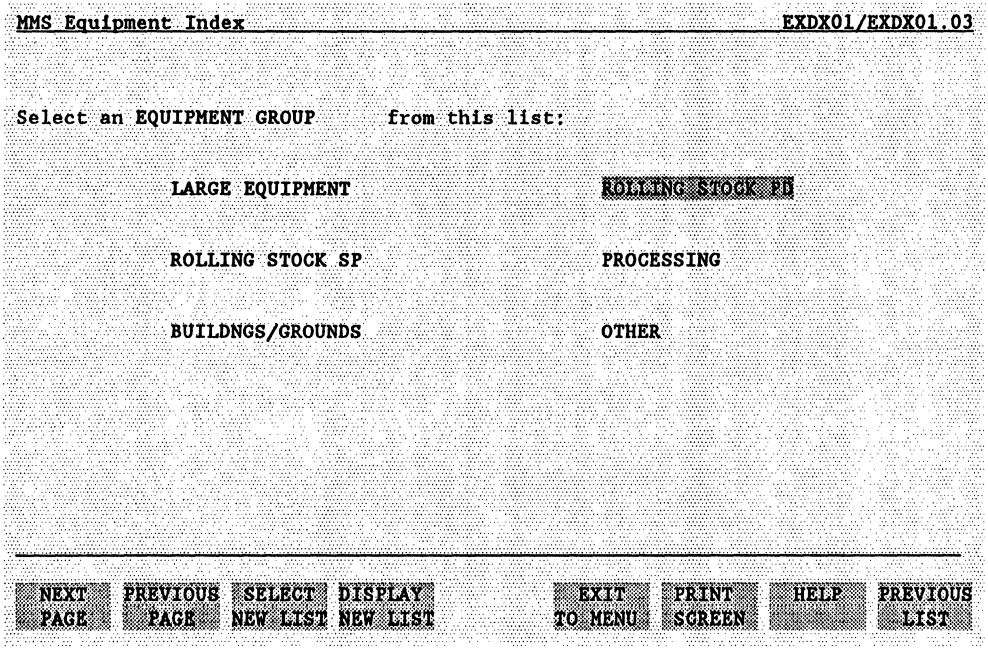
Screen 1 - Equipment Subset Selection



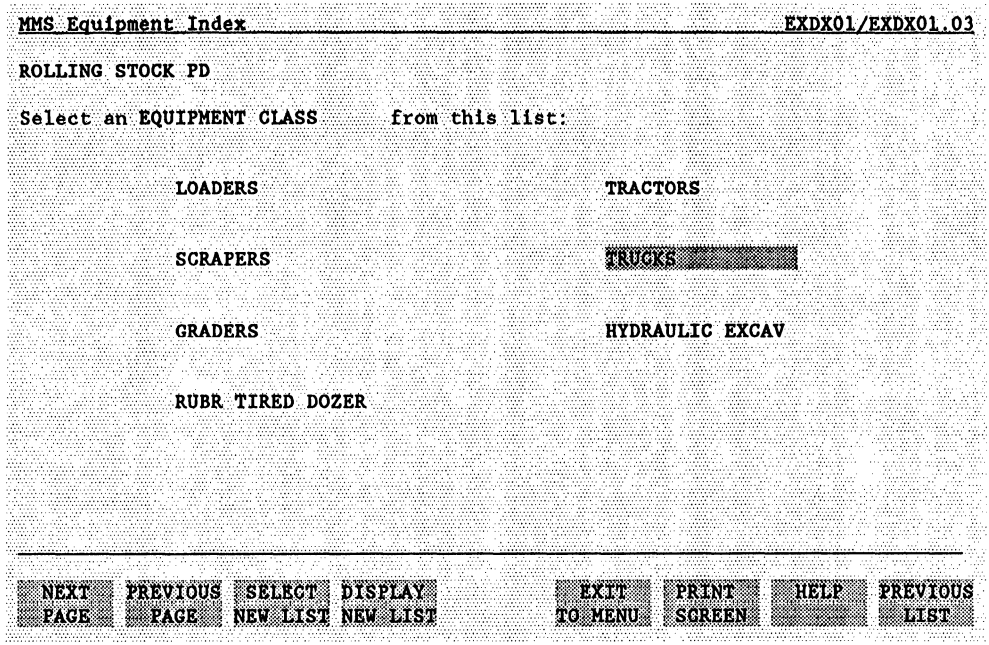
Screen 2 - Subset Selection after touch by the user, proceeds to screen 3



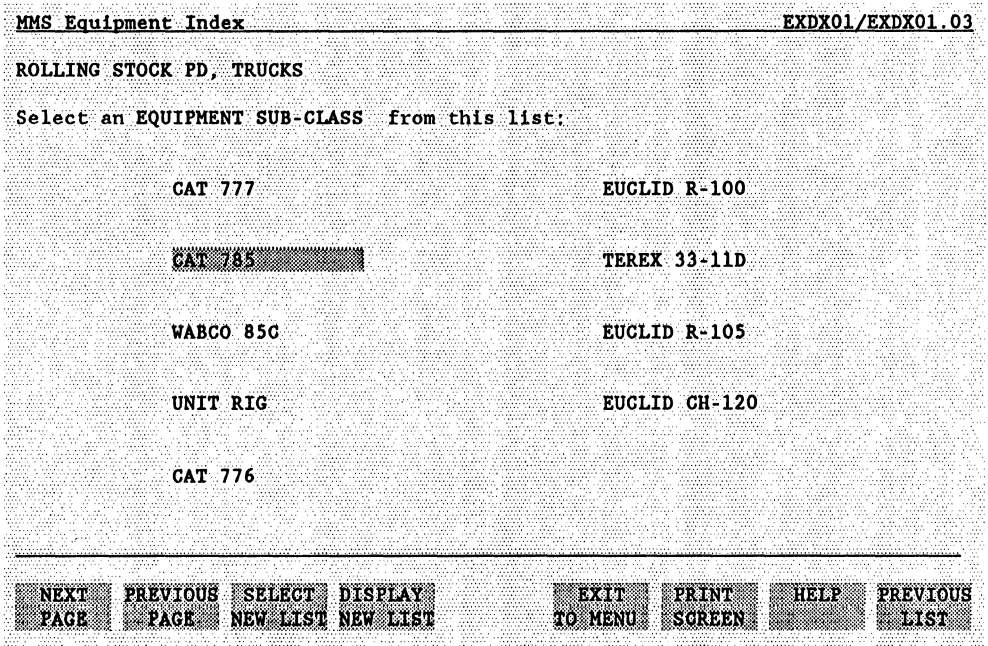
Screen 3 - Equipment Index Selection before touch by the user



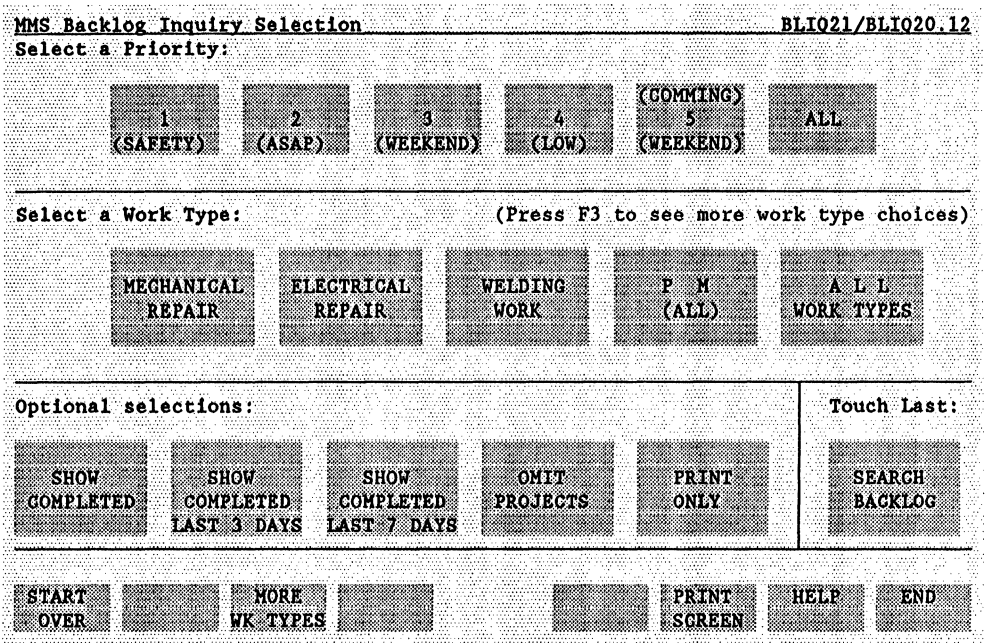
Screen 4 - Index selection after touch by the user, proceeds to next level



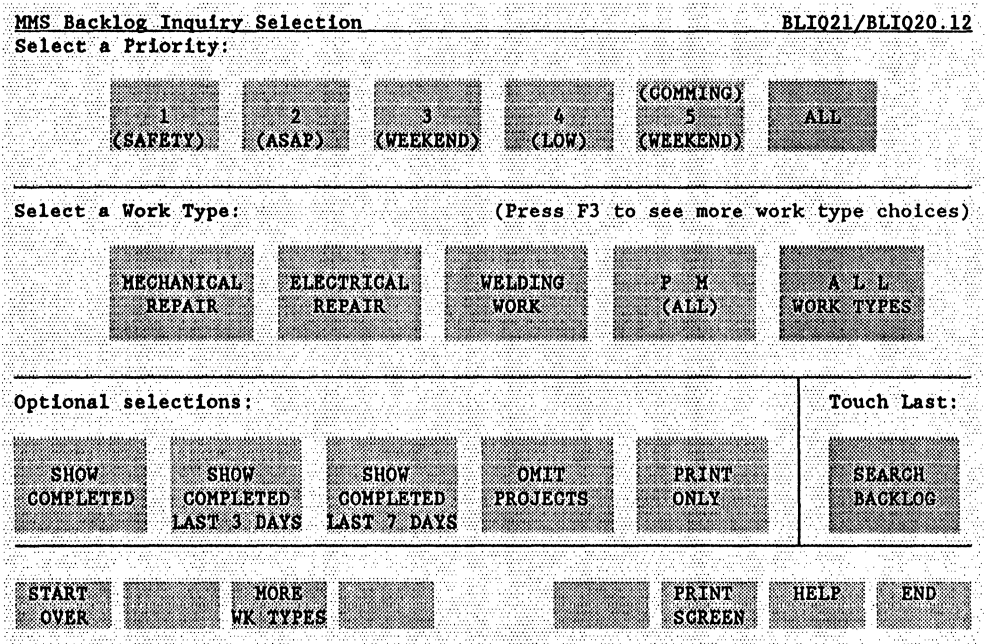
Screen 5 - User selects class TRUCKS by touch, proceeds to next lower level



Screen 6 - User selects sub class CAT 785, proceeds to Option Select screen



Screen 7 - Backlog Search Options Selection (before touch)



Screen 8 - User has touched desired options and started the search

MMS Backlog Inquiry - Generic Format BLIQ03/BLIQ03.02

Group: ROLLING STOCK PD Class: TRUCKS Sub-Class: CAT 785

Unit	Bl#	Entry-Dt	Ty	Description of Request	St	Emp	Pri	Sched-Dt	EstMH	Pts
3033	8406	09/23/86D	PM	FRONT WHEEL BEARING CHE	OP	852	2	03/26/86D		NC
3033	15146	03/10/87G	M	ENG QUITS WHEN PULLING	OP	1492	2			NC
3033	16001	03/27/87D	ME	TRANS CONTROL	OP	632	2			OK
3034	15119	03/09/87D	PM	FRONT WHEEL BEARINGS CH	OP	1334	3	03/08/87D		NC
3034	16239	04/02/87S	M	WEAK HOIST(SLOW)	OP	1035	3			NC
3035	14647	02/27/87D	M	MUD FLAP	OP	110	2			NC
3035	15791	03/24/87G	M	CATWALK BRACE BOLT GONE	OP	1552	2			NC
3035	14445	02/24/87D	M	EXH LEAK	OP	622	3			NC

NEXT PAGE | PREVIOUS PAGE | SELECT ENTRY | DISPLAY ENTRY | PRINT BL LIST | PRINT SCREEN | HELP | END INQUIRY

Screen 9 - Summary list of work backlog for CAT 785 trucks returned by search

MMS Backlog Inquiry - Generic Format BLIQ03/BLIQ03.02

Group: ROLLING STOCK PD Class: TRUCKS Sub-Class: CAT 785

Unit	Bl#	Entry-Dt	Ty	Description of Request	St	Emp	Pri	Sched-Dt	EstMH	Pts
3033	8406	09/23/86D	PM	FRONT WHEEL BEARING CHE	OP	852	2	03/26/86D		NC
3033	15146	03/10/87G	M	ENG QUITS WHEN PULLING	OP	1492	2			NC
3033	16001	03/27/87D	ME	TRANS CONTROL	OP	632	2			OK
3034	15119	03/09/87D	PM	FRONT WHEEL BEARINGS CH	OP	1334	3	03/08/87D		NC
3034	16239	04/02/87S	M	WEAK HOIST(SLOW)	OP	1035	3			NC
3035	14647	02/27/87D	M	MUD FLAP	OP	110	2			NC
3035	15791	03/24/87G	M	CATWALK BRACE BOLT GONE	OP	1552	2			NC
3035	14445	02/24/87D	M	EXH LEAK	OP	622	3			NC

NEXT PAGE | PREVIOUS PAGE | SELECT ENTRY | DISPLAY ENTRY | PRINT BL LIST | PRINT SCREEN | HELP | END INQUIRY

Screen 10 - User has touched a work request to see more detailed information

```
MMS Backlog Inquiry - Detailed Format                                    BLIQ05/BLIQ05.07
                Unit                                  Serial Numbers     Class/Sub-Class
3033  CAT 785 END DUMP TRUCK                          8GB156             TRUCKS
      Current status: UP                                                 CAT 785

Bl#:   8406    Status: OP    Work Type: PM    Priority: 2     Part Status: NC

Short Desc: FRONT WHEEL BEARING CHE
 Long Desc: FRONT WHEEL BEARING CHECK.
            Interval Type=EH, ELAPSED HOURS,  1000 (HOURS)
            Last Done: 01/23/86-D,      1(HOURS)
            PM automatically scheduled by PMUP03
                                                      Reqby:  852 FARLEY
Compnt: 2.6 AXLES              - WHEELS & TIRES       Cmpby:

Est Downtm:          Est labr:          Est Lb Cst:          Est Prt Cst:

Job Master:                                                Date Entered: 09/23/86-D
                                                             Scheduled: 03/26/86-D
Charge Act:                                                    Started:          -
                                                             Completed:          -
   Project:                                       Seq#:

 NEXT   PREV   SEE JOB  UPDATE          PRINT   PRINT   HELP   END
 ENTRY  ENTRY  MASTER   ENTRY           BL ENTRY SCREEN
```

Screen 11 - Work request detail

PRODUCTION GRAPHICS ON THE HP3000 - IT CAN AND SHOULD BE DONE.

by

Steven Carnegie, Richard Corn, and Robert Mattson
Washington Irrigation and Development Co.
1015 Big Hanaford Road
Centralia, WA. 98531

Hewlett Packard has publicly stated that "graphics should be done on micros and not on the HP3000". There are, however, many good reasons for not giving up on the HP3000 as a graphics machine. This talk explores the nature and reasons for business "production" graphics. It then details the success of one group in using the HP3000 as a production business graphics processor. Finally, the methodology, including the use of the contributed graphic library, custom graphics programs and standard hardware is described. The success of the system developed shows that production graphics can be made available in hard copy and online on the HP3000 without "killing" the computer.

OUR MOTIVATION

We were faced with a challenge we expect is similar to that faced by many people and organizations today. We were developing and would be implementing a complex new computer application system which required extensive "online-adhoc" and "production" graphics capabilities. "Online-Adhoc" graphics is graphs produced in "while you wait" mode. This capability calls for as fast of graph production and screen painting as possible. Further, it requires the capability to present the user with as "user friendly" of interface as possible. "Production" graphics, as we are using it, consists of two concepts. One is that graphics are a part of our business operations everyday work. Thus a production graph is similar to a daily or adhoc report or online screen someone might look at in doing their everyday work. Additionally, production graphics implies the capability to produce a lot of graphs with little or no human intervention. We were told by HP that we should be planning on using micro computer technology to do our graphics. HP's strategy did not seem to point the way to a practical or cost effective solution given our requirements and environment. Our solution, now in place and operating quite successfully, involves delivering the required graphics capabilities using the HP3000. This article/talk outlines the who, why, what and how of our graphics solution. Our hope is that it provides inspiration and direction for those of you faced with similar challenges.

OUR ENVIRONMENT, SITUATION AND GOAL

Our company is a large open pit coal mining company located in the state of Washingtion. We are in many ways in the "low-tech" business. The

Information Systems Department supports a full gambit of business related systems from General Ledger to Payroll. More recently our department (7-8 persons) has been involved in a major project to implement a automated Maintenance Management System. Part of the plan for maximizing the benefits from this involved producing online and production graphs. These graphs would be used by the maintenance department supervisors and managers to analyze trends and monitor key variables.

OUR REQUIREMENT LIST

We came up with a list of key requirements for the desired graphics capabilities. Here are the primary ones:

1) Graphs were to utilize extensive multi-year database on the HP3000.

2) Users to have "on-screen" graphics as well has hard copy.

3) Some graphs had to allow for user selection parameter input.

4) Output to HP Laserjets (our main remote printer device).

5) User interface methodology must be simple.

6) User interface to be modifiable to match application.

7) User must be able to produce graphs with minimal training.

8) Online prime time adhoc graphs must be possible.

9) Load/impact on the HP3000 system must be minimized.

10) Rasterizing (hard copy) must be possible during "prime" computer time.

11) Must provide system developers with simple programmatic interface.

12) Must provide graphic system capabilities at low cost.

13) Must be integratible within our programming environment.

14) Provide graphics to all existing online terminal at the same time.

15) Provide graphics availability to users without micro expertise.

OUR SOLUTION STRATEGY

We rejected HP's micro strategy because it didn't meet many of the requirements listed above. Rather what we chose to do is use the HP3000 as the basis for our online and production graphics. We were not able to find any commercial product which would meet all our requirements. We were faced with developing a custom graphics system which would meet our needs.

This posed a number of challenges. But we were able to achieve all our goals by a "little" custom programming. The result of our effort is a system based on the following key pieces:

1) A custom "Graphics Engine" that produces different types of needed graphs based on our "standard" parameters.

2) The capability to produce and store both escape sequence and figure files.

3) A "Rasterizing" program that can output graphs to our HP LaserJets

4) A "touch screen" compatible interface for our graphics capabilities to integrate it in our new system.

5) The "Graphics Initiator" application software for building the parameter file.

6) A "Graph Library" subsystem to handle the storing, cataloging and retrieving of previously created graphs.

What follows is a discussion of the above key pieces in more detail. It may be helpful to follow along on Exhibit 1 which is a block diagram of the key process and interfaces involved in this graphics system.

THE GRAPHICS SUBSYSTEM

Our design criteria called for a graphics subsystem. This subsystem would be used by many programmers to fulfill all their graphic needs with a minimal learning curve about how to write a graphics program. This subsystem consisted of two parts:

Graphics Engine
LaserJet Rasterizer

Both parts are programmed in pascal.

THE GRAPHICS ENGINE

At the heart of our graphics subsystem lies the graphics engine. The engine is a small, compact, processor which recieves data and produces a meaningful graph. The beauty of such an engine is that it hides much of the detail and dirty work of creating a graph. The user of a graphics engine doesn't need to how to draw a line, or plot a point. All he needs to know is what he wants the graph to look like and how to tell that to the engine. From this graph "description" and accompanying data, the graphics engine produces a complete graph.

One of our goals in creating a graphics engine was to produce a tool which any member of the programming staff could use. We studied several types of graphs, found what they shared in common and what was special about each one. From this we decided what type of graphs we wanted our engine to be able to produce. For example, ours creates bar, clustered bar, stacked bar, and line graphs. Knowing what we wanted our engine to do, we designed a programmer-to-engine interface which was "rich" enough in capabilities to allow for all the required actions of the engine, but simple enough so that our programmers could use it easily.

The interface we use is called the graph parameter file. It contains all the commands needed to contol the graphics engine. The command range from "FORMAT LINE5" which tells the engine to make a line graph with five lines to "TITLE1 Sample Title " which tells the engine the main title is "Sample Title." Using this interface allowed us great flexibility in terms of adding new commands to our engine. The engine also receives a data file. This file contains the actual data which is to be graphed. From these two files the engine can produce a graph in two forms, a CGL figure file or terminal escape codes.

FIGURE FILES AND ESCAPE FILES

We initially had our graphics engine using CGL (Contribute Graphics Library) routines to create the graph but it proved too slow. We decided to have the engine send escape codes to our terminals. For our HP150 terminals there are escape sequence codes such as Draw and Move that mimic most of the CGL routines. These escape codes are saved in a file. The file can then be sent to the terminal or saved for later retrieval. We still have the engine use CGL to produce figure files. These figure files can be rasterized immediately or, like the escape files, saved in a library.

THE RASTERIZER

One problem we had in producing a hard-copy of a graph was speed. Using the standard HP rasterizing IFS routines took 10 minutes and slowed the system considerably. The solution was to write our own raster program. "Rasterizing" is a process of taking the "drawing" commands and creating a bit map representation. The LaserJet only understands "characters" and "bit maps." The rastering program reads a figure file and produces a file to send to a LaserJet. Remember, the figure file came from the graphics engine. The file the rastering program produces also contains special codes to command a LaserJet to produce a graph. This file's layout is first a few special codes, followed by a bit map of the graph, and finally some finishing special codes.

CREATING A BIT MAP

The creation of a bit map deserves some description. The first step in creating a bit map is to read, dissect, and understand a "figure file." "Figure files" are the device independant description files created and used by all the Hewlett Packerd HP3000 graphic products. At the time we began, there was no documentation for what a figure file contained. We started from scratch and eventually came to understand what makes up a figure file. Since then, we have seen at least two articles on the contents of a figure file. For the most part there is a one-to-one relationship between CGL calls and commands stored in the file. Each command has a certain number representing it in the file. When you spot the number, then you know the other numbers which follow it are arguments for that specific command. At this point we can read a figure file and pick out the commands: Draw, Move, Text, Arc, LineStyle. The next step in creating a bit map is to process each command and one by one turn on the required pixels. For example, a DRAW command must find all the bits between point A and point B and set them on. This task did not prove too difficult as there are books about such rastering techniques.

The one problem we did have was with characters. The figure file may contain a command such as TEXT "main title" which means write the text "main title" at the current location. But we did not want to take the time to map the characters into bit map representation. That task may be large and we found a different solution. We used the provided character set in landscape mode. The disadvantage to this approach was that we did not have different styles or sizes. But the good part was the speed. Using this rastering process we can produce a complex graph in under 60 seconds. The new raster program is 10 times as fast as the Hewlett Packard process and much less CPU intensive. We are now exploring the possibility of downloading fonts to the LaserJet to allow for different fonts and sizes.

APPLICATION INTERFACE

To utilize the graphics engine in a business specific application system we developed an interface set of calls and subsystem programs. The set that we developed consists of four main components:

Graphics Initiator Programs
Graphics Data Extract Programs
Generic Graph Creation Control Subroutine
Graphs Library Sub-system

The hardware environment consists of HP Touch Screen terminals and LaserJet printers. The entire application interface is programmed in COBOL using VPlus Block Mode screens.

GRAPHICS INITIATOR PROGRAMS

To request the creation of a graph, the programmer creates an application that executes a 'Graphics Initiator' program. This program uses an interactive screen to prompt the user for the parameters needed to produce the desired graph. These parameters are basically application dependent affecting the labeling of the graph and what data will be graphed. When the user has completed entry of these parameters and they pass validation, the program proceeds to graph creation. The initiator program first creates the graphics engine parameter list file. All of the parameters necessary to define the format of the graph are written into this file. The initiator next creates the data extract program's control file. This file contains the user parameters that affect the data extracted. When these two control files have been built, the initiator then passes control to the Generic Graphics Creation Process by calling the Graph Creation Control subroutine. The initiator passes the names of the parameter files and the name of the data extract program to the subroutine. See the example of a Graph Initiator Screen. (Exhibit 2)

GRAPHICS DATA EXTRACT PROGRAMS

These programs read the user data base to extract the data to be graphed and write this data to a data file in a format consistent with the graph to be produced and graphics engine standards. The program reads parameters needed for data selection from $STDIN. When the extract is run in batch or process handled, the parameters are read from the data extract control file.

GENERIC GRAPH CREATION CONTROL SUBROUTINE

The Graph Creation Control subroutine is used to isolate the application system from the actual production of a graph from graph parameters and extracted data. The subroutine begins by displaying a touch sensitive screen that allows the user to select the display medium for the graph being produced. Once the user has selected a display option, graph production begins. The display options are:

Display graph on the terminal screen
Print Graph on a LaserJet printer via a job stream
Save the Graph in the Graphics Libary Sub-system for later review

See the example of the Generic Graphics Control Screen. (Exhibit 3)

DISPLAYING GRAPHS ON THE TERMINAL SCREEN

To produce a graph for on-line display, the GRAFCNTL subroutine process handles the graphics data extract program. This program processes the user data per parameters found in the extract control file and produces the extracted data file. When the data has been extracted, the subroutine displays a message to the user to update the progress of the process and generates a 'beep'. The subroutine then process handles the graphics engine to create the graph based on the graphics engine parameter list file and the newly created data file. The graph is written to the terminal with graphics

display off so that the graph display options select screen remains on the terminal until the graph is ready. When the engine is finished, the Graph Creation Control subroutine toggles alpha display off and graphics display on making the graph appear to the user in a 'flash'.

At this point, the subroutine waits on a Vplus read for the user to press any function key or touch the screen to indicate that the user is finished looking at the graph. If the user presses function key 6, the subroutine will dump the graph to an attached printer (if present). When the user is done, the subroutine toggles the displays so that the display option select screen reappears. If the user touches the 'display on screen box' again, the routine re-displays the graph by toggling the displays. The user can re-display the graph, request it be printed on a LaserJet, save the graph to the Graphics Library, produce another graph with different data or to exit to the menu. See the example of a graph dumped to an attached printer (Exhibit 4). This is also how it looks on the terminal screen.

PRINTING GRAPHS ON A LASER PRINTER

If the user selects the "PRINT ON LASERJET" option, the subroutine saves all of the parameter files and then streams a standard job to produce the graph in batch. If the user had already extracted the data on-line, the data file is saved and the job does not have to re-extract. When the graphics engine is executed in the job stream, it's output goes to a figure file named in the graphics engine parameter list file. When the engine is done, the job stream executes the LaserJet Raster process and creates a spoolfile that is directed to the requesting user's assigned LaserJet. Finally, the job purges all work files. It should be noted that the Graph Creation Control subroutine uses a job streaming routine that performs parameter substitution allowing a single 'pattern' job stream to be used for all graph runs by substituting the file names and other parameters into the pattern. (See the example of a graph printed on a LaserJet. Exhibit 5)

GRAPHICS LIBRARY SUB-SYSTEM

The Graphics Library allows a user or batch process to create a graph and then save that graph for later review. When a graph is created, the user may elect to save the produced graph in the library. If the graph is being printed on a laserJet, the save process is performed in the generic graph print job stream. If the graph is only being saved, a special graph save job stream is run. In either case, the Graph Creation Control subroutine creates another work file that contains save parameters for use by the save process. When a save to the library is requested, the graphics engine outputs the graph to a figure file and to an MPE file that contains the escape sequences and data needed to recreate the graph on a terminal. These two files are saved permanently and the save process updates a directory in the application data base that identifies the graph, the general extract parameters, the date and time of creation, the creator of the graph and the names of the figure and escape sequence files. Finally,

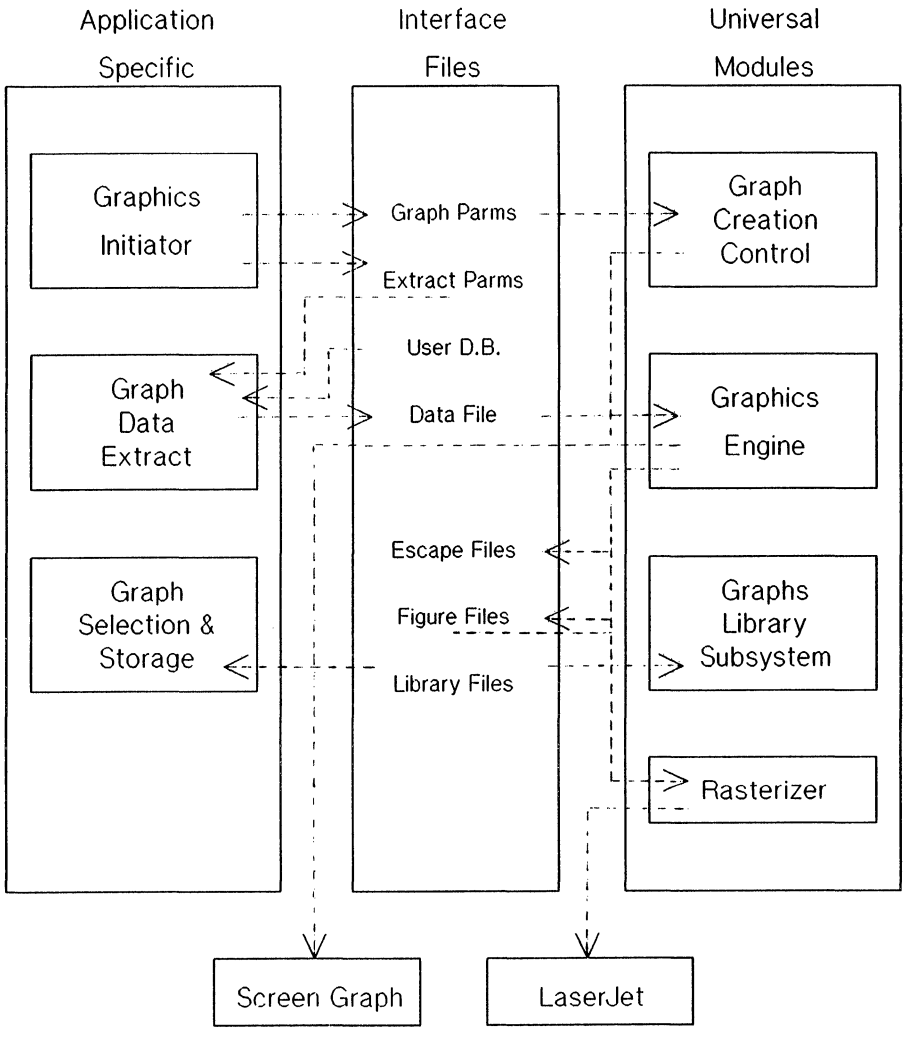
an on-line program is provided that will display a menu of saved graph titles that the user may select from. The selection is made by touching the desired title and the user may then select "display on terminal" or "print on LaserJet." To display the graph on the terminal, the escape sequence file is copied to the terminal. To print the graph, the figure file is rastered to a LaserJet in a job stream.

CONCLUSION

While this system may seem complex and to have many parts, the view that the user has is simple and clean. The entire process runs very quickly although total time will vary with the amount of data extracted. It is not uncommon to examine large amounts of data, graph the results and show it to the user in a minute or less. It is practical enough that users may sit at a terminal creating graphs in an interative fashion to explore and interpret the detail data stored in the data base.

Our online and production graphics system has been in place for almost six months. It is working beyond our most optimistic expectations. The HP3000 has shown itself to be a good machine for producing our needed graphs. We believe there are many reasons why it would not make good business sense to go the micro route for this type of graphics. We hope our experiences help show others faced with the same challenges that there is a way to do production graphics on the HP3000.

WIDCO'S PRODUCTION GRAPHICS



MMS Multi Unit/Sub-Class 12-Month Cost/Op Hr Graph Initiator EMGI23/EMGI23.01

This program produces the **Multi-Unit/Sub-Class 12-Month Comparison** Graph. This graph can be done for up to 3 units or sub-classes. You may select Year, Cost Type and Graph Range. Touch is not used on this graph. You must type the information in the boxes. You may display the graph on your terminal or print it out on a Laser Jet.

Type up to 3 Unit numbers or : 3011
Sub-Class names 3012
3013

Type the year to graph: 1986 (YYYY, defaults to current year)

Cost Type: T (M=mech, E=elec, O=oper, T=Total)

Graph Range: 0 - 100 (Must be a multiple of 10)

PRINT SCREEN | HELP | EXIT TO MENU

Exhibit #2

Example Graph Initiator screen. User enters selections and presses ENTER

MMS Graphics Inquiry Controller GRAFCNTL/GRFCNT.02

Graph Title: 1 YEAR REPAIR COST/OPER HOUR COMPARISON

Select the desired output for your graph:

DISPLAY ON SCREEN | PRINT ON LASER JET | SAVE GRAPH IN LIBRARY | PRINT & SAVE IN LIBRARY

Select the access to your graph if you are saving it in the Graphics Library:

PUBLIC ACCESS | PRIVATE ACCESS

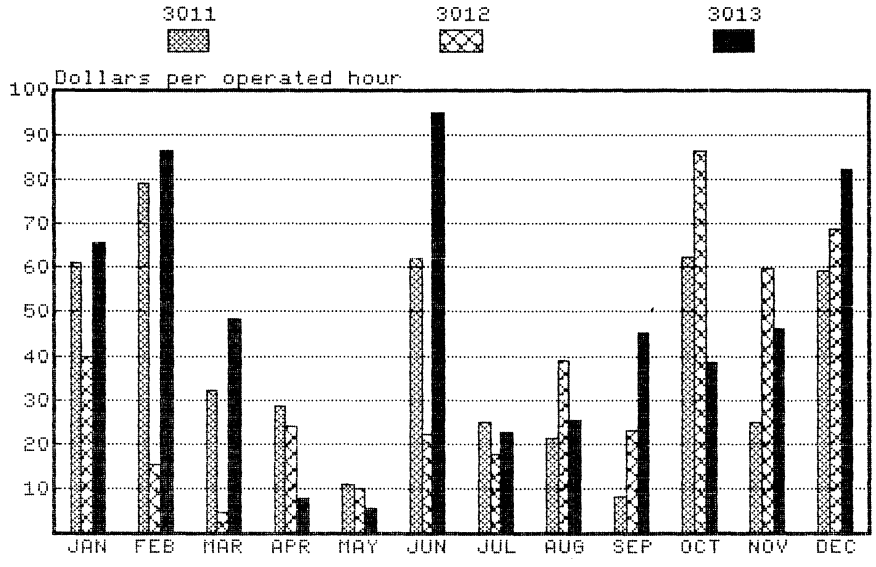
START OVER | CREATE GRAPH (touch last) Job#:

PRINT SCREEN | HELP | EXIT TO MENU

Exhibit #3

Generic Graphics Control Screen - user touches box to select output medium

Widco Maintenance Management System FRI, MAY 1, 1987, 10:45 AM
EMGI23.01 REPAIR COST PER OPERATED HOUR COMPARISON
for Total Repairs



For the year 1986

VER 1.3

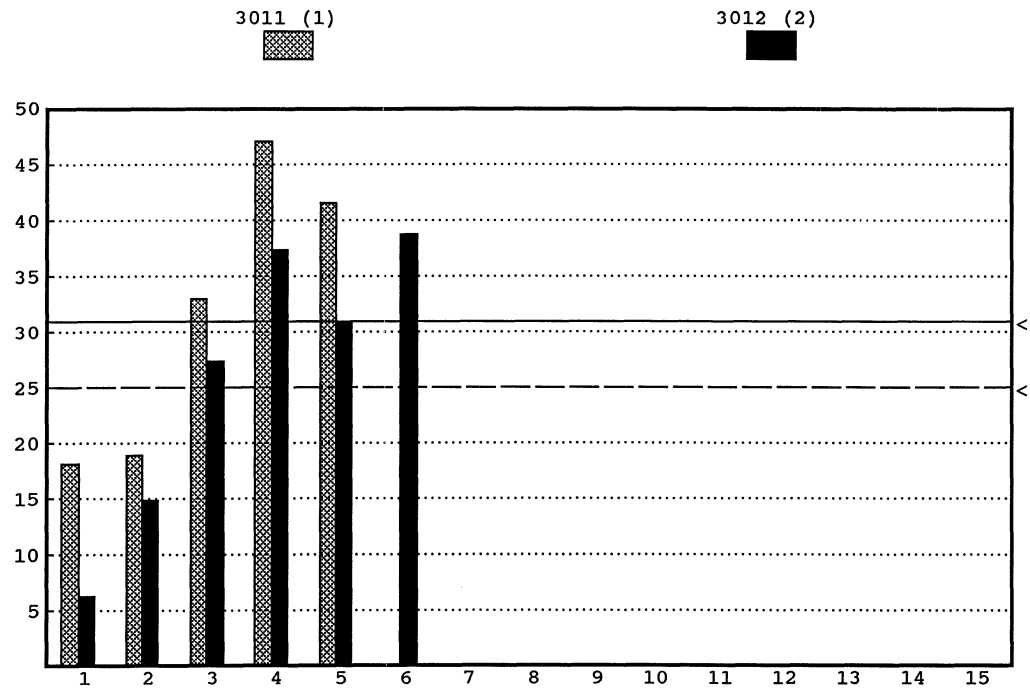
Exhibit #4

Example of screen dump to attached printer of the screen itself.

Widco Maintenance Management System FRI, MAY 1, 1987, 10:54 AM

EMGI31.00 OPERATING HOUR INTERVAL ANALYSIS COMPARISON

Repair Cost Per Operating Hour



< The average of all intervals shown on the graph

periods 1-15, hours 0-37500, intervals are 2500 hrs

Created by: RICHARD CORN VER 1.3

Production Graphics on the HP3000 - It Can and Should be done.
12

Example of one type of Graph produced with our system and printed on an HP Laserjet.

Exhibit #5

HUMAN COMMUNICATION - WHY IS IT SUCH A CHALLENGE FOR SYSTEMS PEOPLE ?

by

Robert R. Mattson
9545 Delphi Road S.W.
Olympia, WA. 98502

ABSTRACT

This thing called "communications" is at the heart of all systems work. Communication is what goes on between systems people and "users", etc. Communication is what we try to do with documentation work, reports ,etc. We sense so often that many problems we face have their root in poor communication. Why? Much of the problem arises from a lack of understanding of key principles of communication, information theory and human limitations. Further, the very nature of "programming" is counter to dealing with communication problems. This talk explores these issues and provides some of the knowledge required to evaluate statements like "But, I told him!".

THE MOTIVATION

The desire to give this talk was based on a series of events happening over a few days time. The capping event was when one of my associates said "But, I told him!" in response to a question about something that had not been done by someone else. It brought to mind a number of recurring thoughts about how little most people seem to understand the fundamentals of "communications". I wanted to respond to my associate but realized that it was more then a five minute conversation. I hope that the concepts presented here help him and perhaps provide you with new knowledge you can apply to your dealings with people on and off the job.

ARE SYSTEMS PEOPLE POOR COMMUNICATORS

First, I'd like to deal with the issue of whether systems people are so bad at communicating. I believe our reputations are justified in many cases. This is unfortunate since I believe we have not understood that the heart of systems work is based firmly on good communication. It is the reason so many systems are not nearly the successes that they could be. In far too many cases, communication is lacking in the analysis, design, development, implementation, enhancement and support of computer systems. The lack of good communication causes numerous problems. These problems ultimately cost organizations money and systems people respect. But,

communication is a two way street. And, since I've thrown stones at my own profession, I'll throw stones at almost everyone else by saying that most other groups are not much better at communicating than systems people. I think we get more press many times because the nature of our deliverables are so "concrete" and visible. Thus when we miss our deliverables due in part to poor communication, it is absolutely obvious to many people. So, if it is true that we are poor communicators and that this is a serious problem, then how do we improve? The first step, after accepting where we are, is to learn more about this concept of " communication". Then, we may be able to view the behavior of ourselves and others in a new light. With this new understanding we may even be ready to make changes in the way we communicate to the benefit of all.

THE DEFINITIONS OF THE WORD COMMUNICATIONS

First, what do I mean by "communication"? The dictionary has a number of definitions. Unfortunately, the primary ones mirror our limited view of what communication should be (1)"an act or instance of transmitting", (2)"information communicated", and (3)"a verbal or written message". These definitions have limitations that I'll discuss in just a minute. However, by digging a little more in the dictionary I did find a definition that comes much closer to what I mean to discuss (4)"to transmit information, thought or feeling so that it is satisfactorily received or understood". One other definition is worth our review: (5) "a process by which information is exchanged between individuals through a common system of symbols, signs, or behavior".

What is wrong with the first three definitions? The first and third imply that it is the action of sending something that is communication. As we will discuss the act of sending is not what is important, rather it is the results. Further, these definitions focus on the sender. What's wrong here? Who should we focus on? The receiver is the most important player in communication...to communicate effectively our focus must be there. The second definition above - "information communicated" - suffers from a circular definition... using the word it is defining to define itself. This is hardly a definition with much chance of increasing ones understanding.

Now let's look at the last two definitions. The key point in definition number four above relates to the "RESULT" of communication. The goal of communicating is to achieve some result...such as an action by someone else or at least that they "satisfactorily...understand" what is being communicated. I am amazed at how much communications goes on which doesn't seem to focus on what is its desired result/goal. The fifth definition is valuable because it focuses on the fact that communications takes place based on "COMMON...symbols (words), signs, or behavior". As we will soon see, getting "common" words isn't all that easy.

THE TROUBLE WITH WORDS

It is the fact that humans can communicate with complex written and verbal languages that has allowed us to achieve our civilization. It is also the same words in these languages that have played a major role in some of mankinds' greatest evils and tragedies. Words are neither good nor bad! But, the fact that most people have a misunderstanding of the nature of words is at the heart of so many problems. What is it that most people don't understand about words and why does this cause so many problems. What most people don't understand is that there is no such thing as a SINGLE definition of any word. Rather, that there are as many definitions to a "word" as there are people who "use" it. Further, even for any single person, a word may change its meaning depending on the context in which the person uses it. If words are so imprecise why do so many people use them as if they were concrete fundamental atoms of communication? Also, we have many words that are used so often that are by their very nature imprecise. How often do we use the words "soon", "important", "cheap", " costly", "immaterial", "later", etc, as if they had some concrete "meaning". It should be obvious that words like these in particular, but actually all words are valuable communication tools only when they are more carefully defined. A word is defined when two people can look at a situation, event, or object and agree that the word, when applied has the same results. An example, I see so often, is when someone says that they will get something done by "tomorrow." Does this mean tomorrow at 12:01 AM tomorrow at 11:59 PM or someplace in between? Apply the same to next week, next month, next year....and the differences in meaning become even more significant. The majority of the words you see, hear, write, read and speak each day can be found to contain this kind of imprecision. No wonder we have poor communication. Where does this lead us? I believe that one way to increase the quality of our communication is to start questioning the meaning of words we "transmit" and those we "receive". Such questioning will cause us to change the words we transmit. It will also cause us to ask clarifying questions of the communications we receive.

THE DOUBLE TROUBLE WITH "SYSTEM WORDS"

Where does this leave us with "system words?" "System words" are all those buzz words which system people are accused of using which the "user" and other people don't understand! You and I know that such words do not have clear definitions even for system people. Therefore, we now need to add another concept important to understanding communications. This concept, which we touched on briefly under definitions, is that communication must involve "common" symbols. As we've discussed, everyday words are not very "common" in their meaning. Technical computer terms are even more misunderstood. Is it any wonder we have problems? The world is rapidly changing related to the use and understanding of computer related technology. However, a large gap still remains between the "systems" world and the "users" world. If we are to communicate effectively...we must either make the meaning of the words we use known to the person(s) to whom

we are communicating or we must find words that are already common between us to use in place of technical words.

THE 5 TO 7 RULE

A concept that is broken everyday in communications is the "5 to 7 rule" from experimental psychology. The psychological research showed that the average person can only hold from five to seven numbers, words, or concepts in their head at one time. When we are asked to deal with more than this range we start to lose some of the numbers, concepts, and/or words. One of the common places this concept is ignored, to the detriment of communications, is in presentations where a list of 10,20,30 or more items are presented for discussion. The average listener will not be able to hold all these in their brain at once. It can cause a type of "swapping" that leads to blank times when communication coming from the speaker is lost. If you look around I'm sure you will see many examples of this rule being broken and the obvious results. When more than seven of something need to be presented, use the concept of "grouping" to reduce the major list to seven or less and have seven or less in each minor group.Use the opportunities that you have to apply this important psychological concept "the 5 to 7 rule."

LEFT BRAIN AND RIGHT BRAIN DOMINANT PEOPLE

Recent studies have indicated that there are major differences in how individuals process information. The studies show that individuals differ in whether they deal best with information communicated by verbal OR visual means. Individuals also differ in whether they interpret lists of facts better than diagrams or pictures. The studies are complete enough for us to garner a key strategy related to our communications....be a multi-media communicator! While we may not know whether a person is a listener, a reader, a list learner, or picture processor; we can maximize the chance of our message being received by using all types of communication methods!

MULTIPLE OCCURENCE LEARNING

I don't know where we got the commonly held belief that people learn by being "exposed" to some concept or fact ONCE. Yet think of the number of times that people do just that - communicate something only once. This situation is epitomized by my associates "I told him" statement. If you really want to communicate something then communicate it more than once. As an aside, if you really want to learn something study it more than once. All the concepts we are discussing apply equally well to a systems person as listener. In otherwords, if we want to increase our understanding of what someone is telling us - then have them tell us multiple times.

COMMUNICATION TRANSMISSION LOSS

Let's discuss the concept of what goes on in communication transmission to explore the loss factor that might be involved. When communicating, the sender puts together a message to be transmitted. I'll suggest that a average communicator only produces a 75% perfect message. In otherwords, the average person doesn't say what they mean exactly most of the time. Now we all realize that we aren't perfect listeners. Let's say we only hear 75% of what is sent. Now we add in the fact that of what we hear, we only interpret correctly to the tune of 75%. If we multiply these factors, we realize about 42% of any communication gets from sender to receiver. I expect that is optimistically high...but it should make the point that human communication is not a 100% transaction. Now I would leave you to wonder....which 58% of the last communicating that you did...was lost in transmission....was this lost information important?

COMMUNICATION LOSS AND "BEING THERE"

If we lose so much in any communication, then we in systems ought to change some common practices. One that comes to mind is the concept that we don't involve key people in system analysis and design. So many times the actual programmer and actual user are not the ones that discuss the problem and the solutions. Rather, the prevalent concept/practice is of a system analyst and a user manager/supervisor supposedly solving problems and making decisions. How much information is lost in transmission. Another, is that we should write more things down. How many times do we have an important conversation with a user....and yet fail to write down a summary and give them the feedback of what we heard. I believe the extra time spent doing this would save frustration, false starts, and redo costs.

SHORT TERM MEMORY VS. LONG TERM MEMORY

Okay, now we know how to communicate at better than 42% accuracy. Now we're facing another human limitation factor. Studies show that the retention of facts appears to involve short term and long term memory mechanisms in the brain. The average person forgets nearly 90% of what was communicated to them with the passage of only 24 hours. The remaining 10% is lost more slowly but only around 2% makes it to our long term memory storage. These studies do deal with many factors which may or may not always be the case in our day to day communications. Nevertheless, even if we optimistically assume 50% retention, it sure doesn't leave much of what we communicated (remember only 42% got through) there in someones head when we discuss it with them a week later. What do we do? Besides writing down what we tell people verbally, we can remember that when we meet with someone to discuss something we talked about two or more days ago that maybe some review is in order...to bring everyone back up to speed and refresh memories as to what was discussed.

MOTIVATION FACTOR IN COMMUNICATIONS

As a friend pointed out when we were discussing some of the issues related to communications...."the receiver has to care!" What this means is that all the best communication techniques in the world may not make a difference in a situation where the receiver doesn't care about the subject being communnicated. I believe this is true and getting them to care is beyond the scope of this article. However, maybe keeping this comment in mind will help us to understand even better that it's not a perfect sender-receiver world. Maybe, in those situations, just keeping this concept in mind will allow you to adjust your techniques and effort to the appropriate level.

SYSTEMS WORK AS IT RELATES TO COMMUNICATIONS

I believe that how well we do in systems work is integrally tied to how well we communicate (send AND receive). It is a paradox however that the work most of us start out doing, teaches and reinforces concepts that cause us problems later on with human communication. Communications in programming involves a very exactly defined vocabulary. The computer interprets the communications to it in exactly the same way each time. There is no loss of information in transmission. The computer will listen with 100% accuracy. The computer will react to a particular transmission in exactly the same way each time. The computer is neutral. It doesn't have bad days or have things on its mind. The computer doesn't care that you caused it to crash yesterday! Everyone of these concepts is almost exactly opposite from what it is to deal with human communication. Thus the early years in this profession, day after day tend to train us in a mind set which if applied later in our careers will cause us many problems. Add to this the fact that almost no systems academic training curriculum includes any practical human communications or psychology classes. The result is not suprising. Most systems people could improve their communication abilities greatly.

UNDERSTANDING IS THE FIRST STEP TO IMPROVEMENT

I believe that understanding is the first step to changing for the better. Systems people are good learners. The facts are there for one to learn. Good quality human communication is not so difficut if we remember the game is played with very different rules than programming. To recap briefly:

1) The receiver is the most important player in communication.

2) Choose and define your words very carefully. Use words "common" to both sender and receiver.

3) Remember the "5 to 7 " rule - ie. that most people can only handle this many items at once.

4) Remember to present your material in a variety of ways - verbal, written, and illustrated.

5) Present your material more than once to update and refresh memories regularly.

6) Understand the motivation of those with which you communicate.

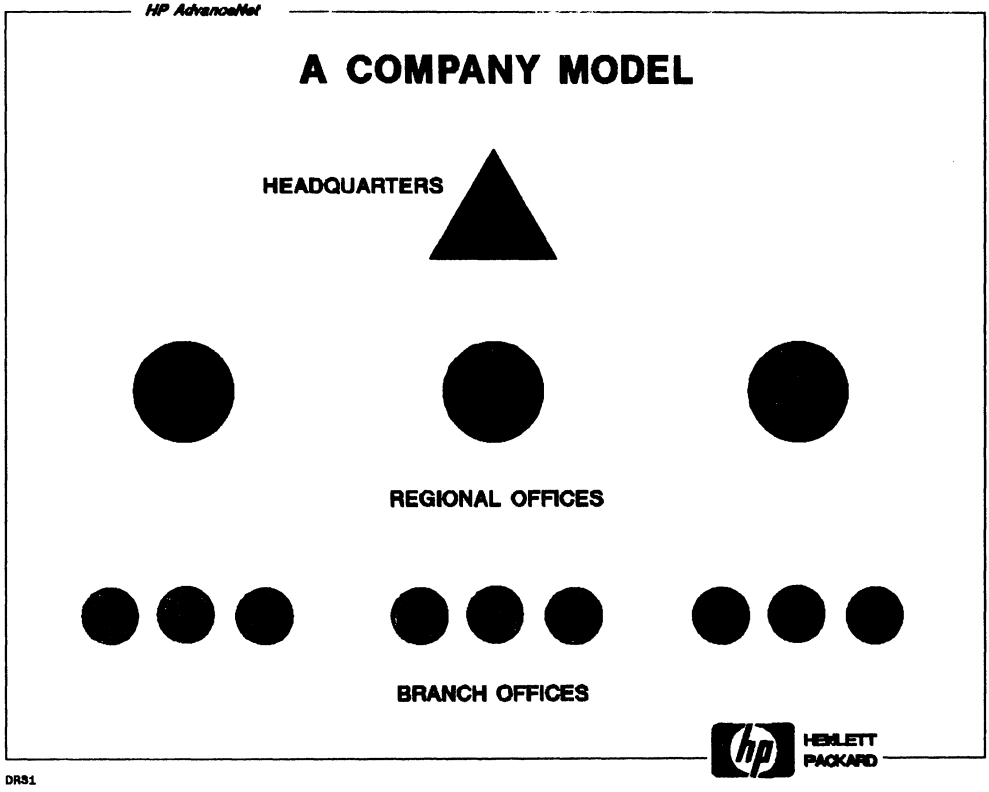
7) Remember the difference between computers and humans.

CONCLUSION

Well, now "you've been told." I hope this little excursion into the area of communications helps us all achieve those many goals which depend on it. Good luck! May you always understand and be understood.

REGIONAL SALES and SERVICE NETWORKING SOLUTION

DOUG MCLEAN



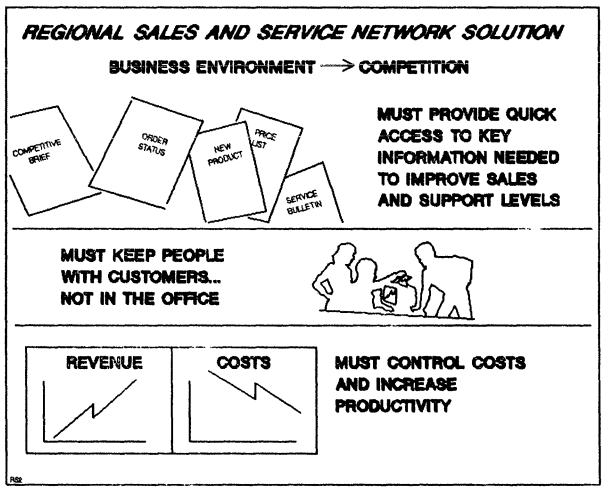
DESCRIPTION OF ENVIRONMENT

It is often a "business" need to have facilities spread over a wide geographical area. This can happen because of economic reasons (lower rents, lower transportation costs, availability of labor or materials, etc), historical reasons (acquisitions, etc) or for market reasons (need to locate your sales offices in the market they serve). Companies which are distributed geographically need a network solution that ties these sites together. For large corporations, this is typically done with a company-wide network solutions. For small to medium sized companies (or for regional "subnetworks" within a large corporation) the Regional network provides this capability.

A major untapped opportunity for automation today is the sales and service function within companies. Sales and service forces are finally being recognized as a key competitive advantage and the benefits of providing computer based tools to increase the effectiveness and productivity of these areas is beginning to capture the attention (and pocketbook) of many major companies. The reason for this is simple. Companies want to get ahead of their competition.

This means getting to your customers first with timely, accurate information. It means liberating field personnel's time from administrative tasks so they can spend time planning, servicing and selling. The typical VP of Sales is under tremendous pressure to make this happen, and this is where HP can help you make the difference.

Companies that are geographically dispersed (or have a sales/service force that is dispersed) have a difficult communications problem to overcome. Information must be dispersed quickly, financial information consolidated and organizational communication such as electronic mail is required to keep on top of the activities needed to make the company grow and prosper. This is the role of the Regional Sales and Service Network. Whether you call these sales offices, parts depots, claims office, or retail stores; whether these are located in one state, one region, one country or an entire continent, the Regional Sales and Service Network can meet your needs.



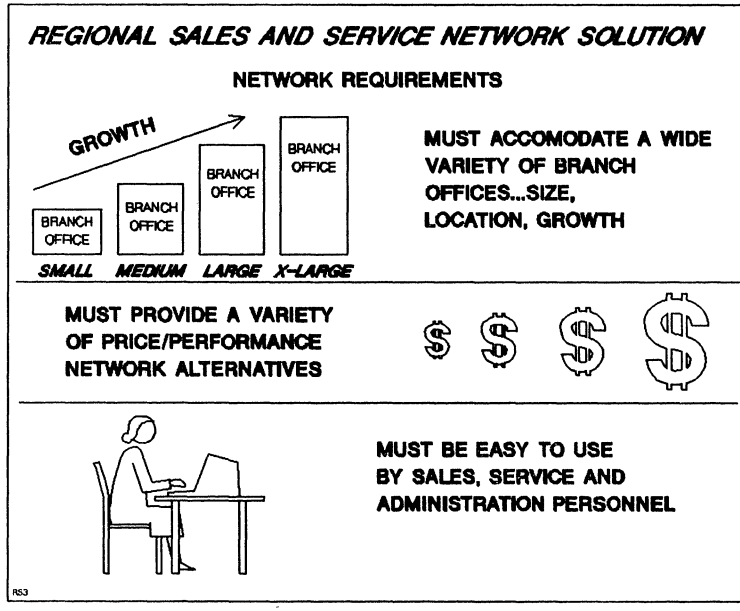
Regional Sales & Service Networking Solution

NETWORK REQUIREMENTS

A Regional Sales and Service force is by its nature distributed. This means networking must be used to connect field personnel and branch offices to their regional headquarters. Most companies have a variety of sizes of branch offices, from an individual person working from their home, to a very large office with hundreds of people. The branch office environment is not static, each branch office will grow and change over time, many will move geographically, and new ones will be added and old ones closed. Thus the network must accommodate this dynamic environment with solutions that fit a variety of sizes and growth rates.

In addition, there are a wide variety of communication options available today, each providing their own price performance trade-offs. The network must accommodate a variety of these communication choices to enable the customer to minimize phone bills yet provide the level of service needed by each branch office.

With the growth and change in this environment, no one can afford to spend lots of time training users on how to use complex, cryptic commands to access the network. This is why HP has made access to other systems and information as transparent to users as possible.



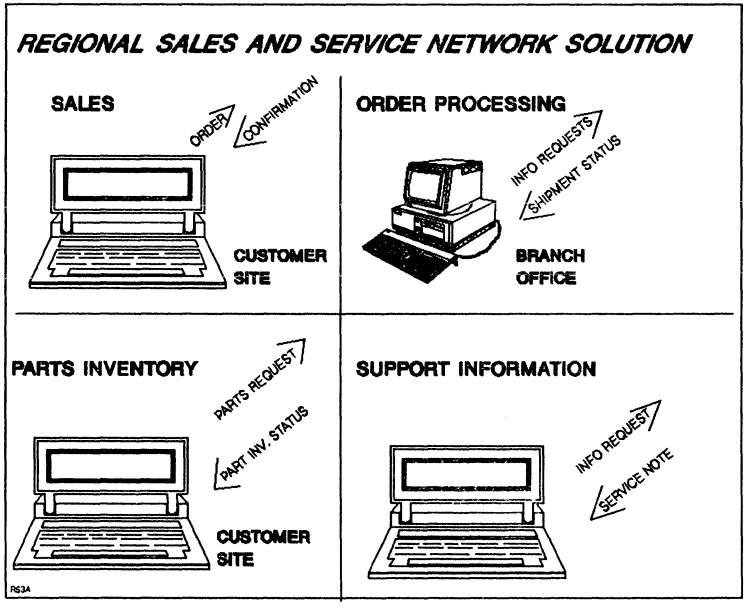
APPLICATION EXAMPLES

A sales rep for a volatile commodity (such as I/Cs or petro chemicals) visits the purchasing agent of a large company. The customer wishes to lock in a price and get an assured delivery date. The sales rep can connect the HP Portable Plus into the regional order processing system, enter the order and get an immediate acknowledgement. This can assure that the customer gets the supply when needed and avoids the embarrassing and time consuming problems, clerical delays and errors that cause shipment delays or price changes. It also assures the selling company that they will get the price they need to make a profit.

A customer calls into the office to find out when an order will be shipped. The order processing department can network to the warehouse to find out the exact status and, if shipped, get the date and waybill number.

A service repair person is on site without the part needed for the repair. Using the Portable Plus computer the repair person can access the service parts inventory system and find out if the part is available at the branch office, another branch office or at the regional headquarters. If available, the part can be processed for immediate shipment to the branch office and the customer notified when the operation will be back up and running.

This same person could then access the latest field notes for the equipment and make sure that any other updates or maintenance that this equipment requires is scheduled and the parts available for the return trip to the customer site to complete the final repair, thereby preventing future downtime and economizing travel time.



Regional Sales & Service Networking Solution

REGIONAL NETWORK

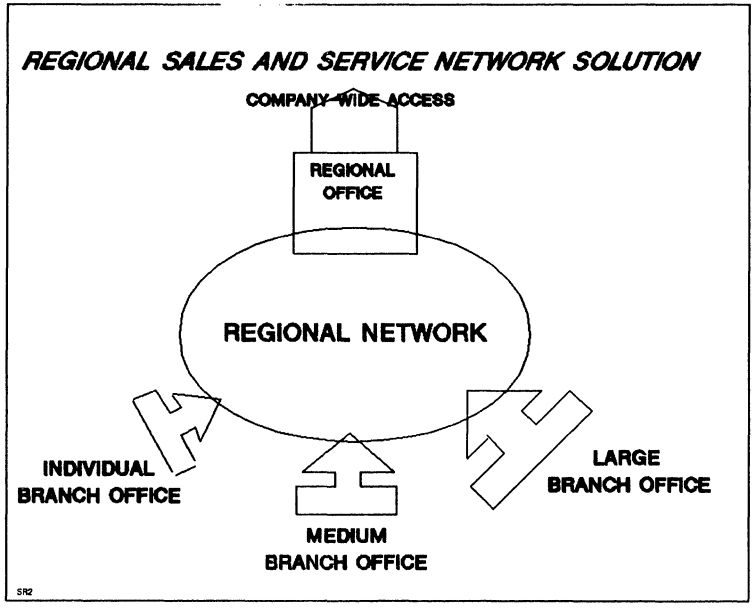
The Regional Sales and Service Network is made up of three major components or modules:

The Regional Network Module
The Branch Office Network Module
The Company-Wide Module

At the heart is the Regional Network Module. This provides the Networking between the various sites and typically represents the highest area for potential savings. Through the proper choice of communication alternatives, monthly operating costs can be minimized while providing the desired service level and performance.

The second major networking module provides for the networking at the branch office. This must accommodate various sizes and types of branch offices from an individual terminal, PC or HP Portable Plus, to a large branch office with hundreds of PCs, terminals and multiple HP3000 systems.

Tying the regional office into the company-wide backbone network is the Company-Wide Access module. It must accommodate your choice of company-wide backbone, whether this is X.25, SNA or point-to-point.



Regional Sales & Service Networking Solution

REGIONAL NETWORK

The Regional Network Module provides the communications alternative needed to connect branch offices together and to the regional office. It provides for the consolidation of inputs and allows for management of the network from the regional site.

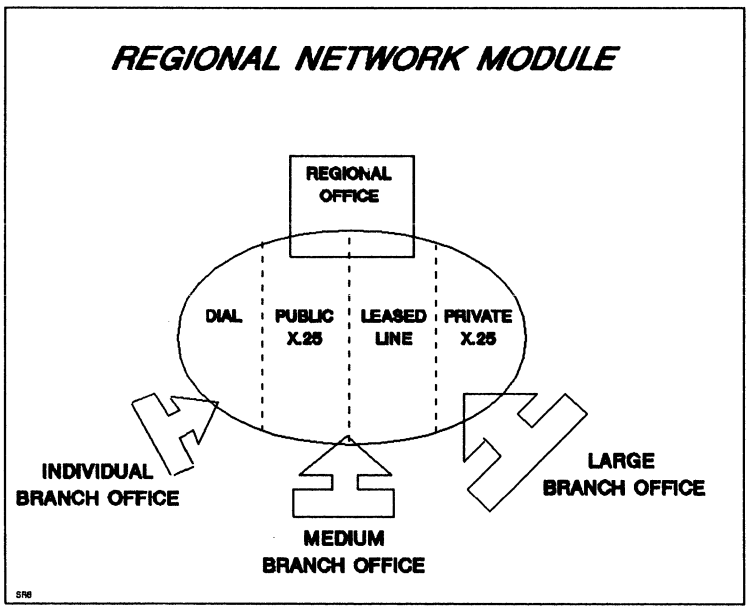
The major communications alternatives offered by the Regional Network Module are: Dial-up phone lines, public X.25, leased phone lines, and private X.25.

Dial offers the lowest up-front investment and, for applications requiring only low volume batch communications, will offer the most economical solution. However, as the communication volume increases or as the need for interactive access to data becomes important (for on-line queries to parts or order databases for example), dial-up may not provide the best solution.

Public X.25 provides an excellent alternative for both batch and interactive application with low to medium data volumes.

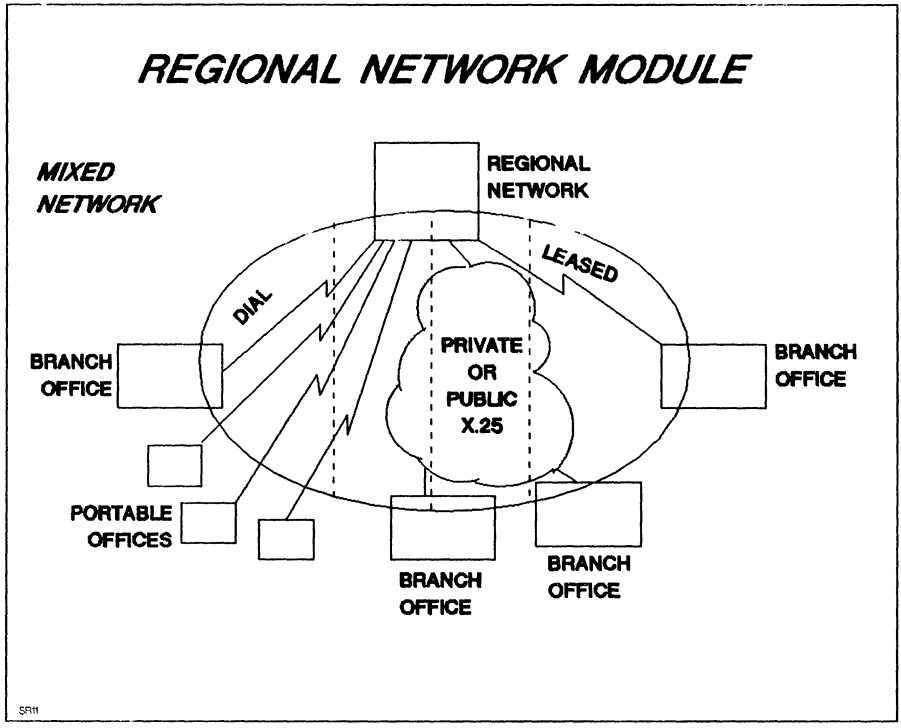
Leased lines are best for higher data volumes and provide both batch and interactive application access.

Private X.25 is best suited for companies that want the maximum security and flexibility in their regional communications.



REGIONAL NETWORK MODULE

A Regional Network will typically use several of the communication alternatives in combination to provide the best price performance for each branch office. Your HP Network Consultant can assist you in choosing the right communication alternatives to fit any particular situation.



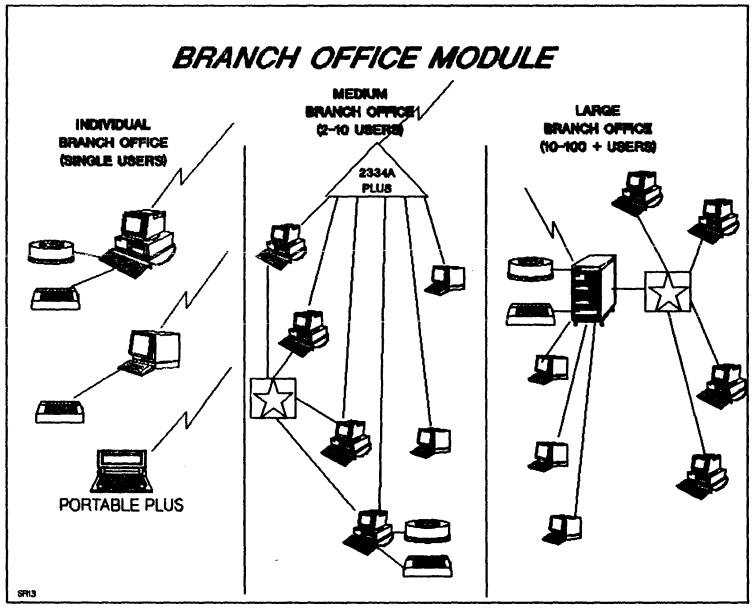
Regional Sales & Service Networking Solution

BRANCH OFFICE ALTERNATIVES

Branch offices come in many sizes and have differing computing needs. The smallest has a single terminal, PC or Portable Plus. The largest has an HP3000 (or multiple HP3000s) based solution. For small single user offices, dial up access using terminal emulation and file transfer capabilities of HP AdvanceLink over RS232 provides access to other branch offices or regional offices. Alternatively, The Serial Link on the Vectra or HP 150 allows remote PCs to access the Personal Productivity Center Service on an HP3000 in the regional office. This PPC solution makes the network transparent to the branch office user and is easy to use for "novice users".

Medium sized branch offices (2-10 users typically) are best served through a multiplexer or "PAD" such as the 2334 Plus. This allows the branch office to share one communication line (dial, leased or X.25). Local resource sharing for PCs can be provided to the medium branch office through the OfficeShare family of PC Local Area Networks. In addition, serial printers can be connected to the 2334 Plus for local printing capability (Note: Today PPC access is not possible in this configuration).

Large branch offices are typically centered around the 3000 providing local office and data processing functions. This 3000 can be connected through dial up access using the ATP to the regional office using the new Serial Network Link. Alternatively, X.25 or leased lines can be used to connect to the regional office.



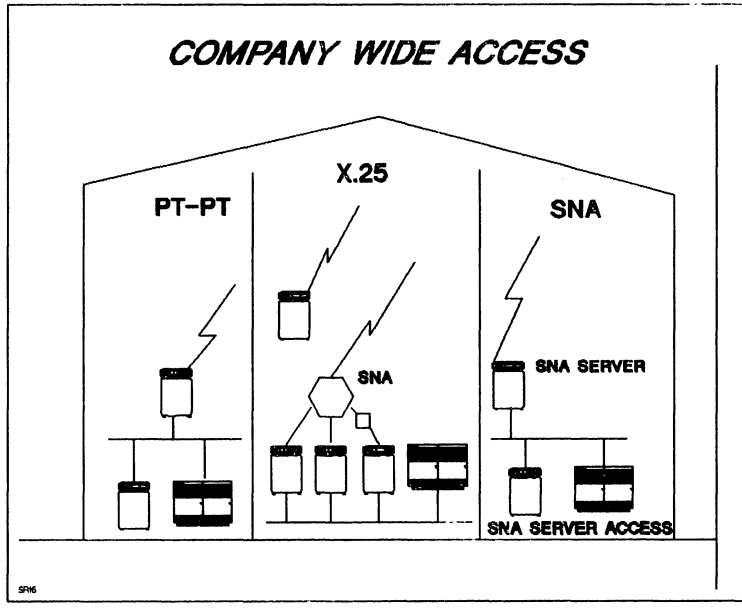
COMPANY WIDE ACCESS

The selection of the company wide access method depends largely upon the company-wide backbone network that is in place. If, for example, your company has selected X.25 for the backbone, then an X.25 access method would be required.

The selection of the company-wide access method depends largely upon the company-wide backbone network that is in place. If, for example your company has selected X.25 for the backbone, then an X.25 access method would be required.

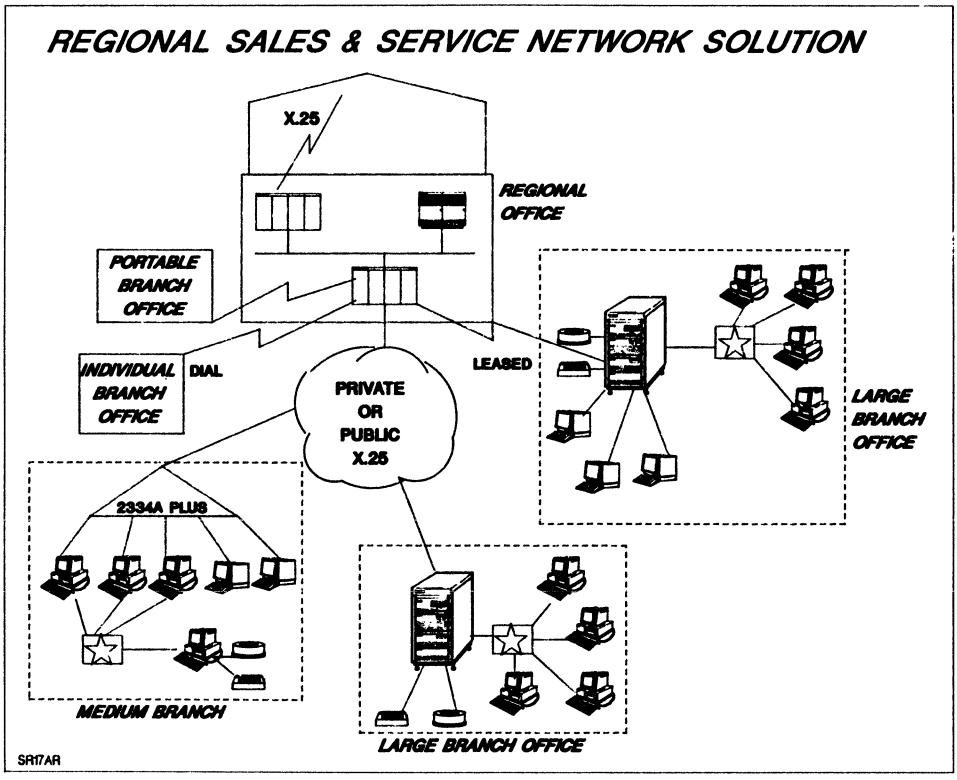
If your company is small and has not selected a company backbone, then either point-to-point or X.25 can be used. If you are connecting 2 or 3 sites together, point-to-point is typically the best alternative. For more than 3 sites, X. 25 will typically be the best solution.

If your company has IBM systems and has selected SNA as the company-wide backbone, then the SNA link may be the lowest cost alternative. However, if multiple 3000 sites need to connect to one company headquarters site, it is often better to place an HP3000 at the headquarters site, connecting this to the IBM system using the SNA link and use this as a "gateway" connecting to the multiple 3000 sites via X.25. Creating this "subnet" within the larger SNA Network will often provide the lowest cost solution.



THE REGIONAL SALES and SERVICE NETWORK SOLUTION

HP has a complete set of products to create any or all of a Regional and Service Network today. Putting together these modules and combining them with the Business Office Network Solution, you can quickly see that today, HP has powerful network solutions for small to medium sized companies which are geographically dispersed, and for the Sales and Service forces within large companies. These products are here today and are meeting customer's needs.



Regional Sales & Service Networking Solution

TWENTY RULES TO BUSINESS GRAPHICS

James E. McLean
JMA, Inc.
P. O. Box 270507
Houston, TX 77277-0507

Business graphics are an extremely useful and flexible medium for explaining, interpreting, and analyzing numerical facts, largely by means of points, lines, areas, and other geometric forms and symbols. Business graphics make possible the presentation of data in a simple, clear, and effective manner. It also facilitates comparison of values, trends, and relationships. Graphs possess certain qualities not found in textual presentation. These values may be summarized as follows:

1. Charts which are designed well are able to create a higher interest and are able to attract the attention of the reader better.

2. Statistical charts provide a clear, effective, and precise method of conveying a message. In many situations, charts are better than the written word. Also, charts and graphs with the visual relationships are more easily understood and remembered.

3. Graphs and charts save time since large amounts of data can be reviewed and retained quickly by the audience.

4. By providing a comprehensive picture, charts and graphs can illustrate problems that will give the reader or audience a more complete and balanced understanding.

5. Graphs can also bring to light hidden facts and relationships which then can aid in developing an analytical solution.

Unfortunately most people quickly understand the usefulness of graphs and charts but lack the basic knowledge of graphic design as stated in the January, 1987 edition of Mini-Micro Systems (p. 67). "All the software in the world won't make you a graphic designer." This quote summarizes the limitations of 99% of business graphics users. To help develop and design graphics, we devised Twenty Rules to Business Graphics. The user's goal with business graphics should be to make certain the graphics he uses (1) addresses his target audience, and (2) is easy to understand and pleasing to the eye.

There are more types of graphs and charts than the ones discussed here, but the ones not covered are seldom used. We will limit the scope of this article to presentation graphics using either 35mm slides or overheads.

The first two rules address the image size of the graph.

RULE 1 - IMAGE SIZE OF 35MM SLIDES

The image size needs to be in a ratio of 3 units horizontal by 2 units vertical, or when dealing with an 8.5" by 11" original piece of paper, the image size needs to be 8.5" horizontal by 5.75" vertical.

RULE 2 - IMAGE SIZE OF OVERHEADS

The image size of an overhead transparency should fit in a 7.5" horizontal by 9.5" vertical area if you use an 8.5" X 11" original layout. For a clear projection you should be able to read the transparency from about ten feet away without a projection light.

A major area of difficulty is deciding which type of chart should be used in a given situation. Rules 3 thru 12 should be able to answer any difficulties you may encounter.

RULE 3 - HORIZONTAL BAR CHARTS

Bar charts are used when you are comparing size or emphasizing the difference between two things. Time is not a factor with bar charts. Limit the number of items in simple bars to six or less. Example: comparing the number of people by division or sales by territory. (See Figure I.)

RULE 4 - COLUMN CHARTS OR VERTICAL BAR CHARTS

Column charts or vertical bar charts are used when comparing the same elements over a period of time; i.e., sales by quarter. Limit the number of items in simple columns to six or less. (See Figure II.)

RULE 5 - STACK OR GROUP BAR AND COLUMN CHARTS

Stack charts are used to compare parts to the total item. (See Figure III.) Grouped charts are used to compare the relationship within an item. (See Figure IV.) An example would be quarterly sales by divisions.

RULE 6 - CONSISTENT SPACING

When using verticle bars or columns, the space between them should be one half the width of the bar or column. When using stack or group bars or columns, the spacing should be one or one and a half times the width of the bar or column.

RULE 7 - PIE CHARTS

Pie charts should be used to show the relationships of parts to the whole. Never use more than six slices. They should be arranged from the largest slice starting at 12 o'clock and moving clockwise to the next biggest slice and finally ending with the smallest slice back at the 12 o'clock position. (See Figure V.)

RULE 8 - LINE CHARTS

Line charts are used when plotting a long series of data with many points. They are used to emphasize movement or change such as monthly sales figures for the entire year.

RULE 9 - LINES IN LINE CHARTS

Lines of the line charts need to be thicker than the axis lines. Try not to have more than five lines in a chart and use contrasting colors. Use distinctively different line patterns when using black and white. (See Figure VI.)

RULE 10 - TEXT CHARTS

Text charts are used only in presentations to highlight or to emphasize major points. Key words should be used, not sentences. The number of lines should not exceed seven nor have more than six or seven words on each line. Idealistically, you should use five lines with four words or less on each line. When using color, make the text all the same color except when you want to emphasize a particular point; then use one line of text in a different color. (See Figure VII.)

RULE 11 - TABLES

Tables should be used when wanting to convey specific information. An example would be units produced by shifts per week for the last four weeks.

RULE 12 - TABLE DESIGN

When presenting tabular information as a graphic, use a minimum number of lines to lay it out. Instead, use spacing to separate facts and keep the number of physical lines to as few as possible. Align all headings and columns containing words to the left. Make certain all numeric data is properly aligned in columns; i.e., all commas and decimal points fall directly one beneath the other. Make certain that any captions used are specific, and the units of measurement are clearly stated. Also, be sure the figures are correct and the totals are accurate.

COLORS

Another area to consider when designing graphics is color selection and coordination. Remember, graphics must be appealing to the eye of the audience and color itself can be used to convey ideas or impressions. Rules 13 thru 20 will assist you in your color selections.

RULE 13 - WHEN IN DOUBT, CHOOSE BLUE.

Blue should be used for basic data, facts, statistics, organization, logic, reality, stability or to establish reference.

RULE 14 - RED STANDS FOR POWER OR DANGER.

Red should be used to alert, show problems or conflicts, debits or over-budget, behind schedule or indicate that which needs immediate attention.

RULE 15 - GREEN IS COOL AND RESTFUL. IT REPRESENTS NATURE.

Green should be used to show action, work production, construction, manufacturing, price performance data, sales, marketing, or financial matters.

RULE 16 - YELLOW IS NEUTRAL AND STANDS FOR THE SUN.

Yellow should be used for assets, investments, profits, projects, research, theory, design, invention, energy, or the future.

RULE 17 - BACKGROUND COLORS

Background colors other than clear are used mainly with text slides. The background should always be light and complement the text color; i.e., blue letters with a light yellow/orange background or red letters with a light blue/green background.

RULE 18 - COLOR COMBINATIONS

Color combinations are best shown with a color wheel. Try to avoid the bright colors; if used, limit them to a small area. It is best to use colors that are subdued. You may want to use two complementary and one adjacent, two alternates and one adjacent, three alternates or three adjacents for color combinations. If you need additional color, you may wish to choose a neutral or another adjacent color.

RULE 19 - BLACK AND WHITE

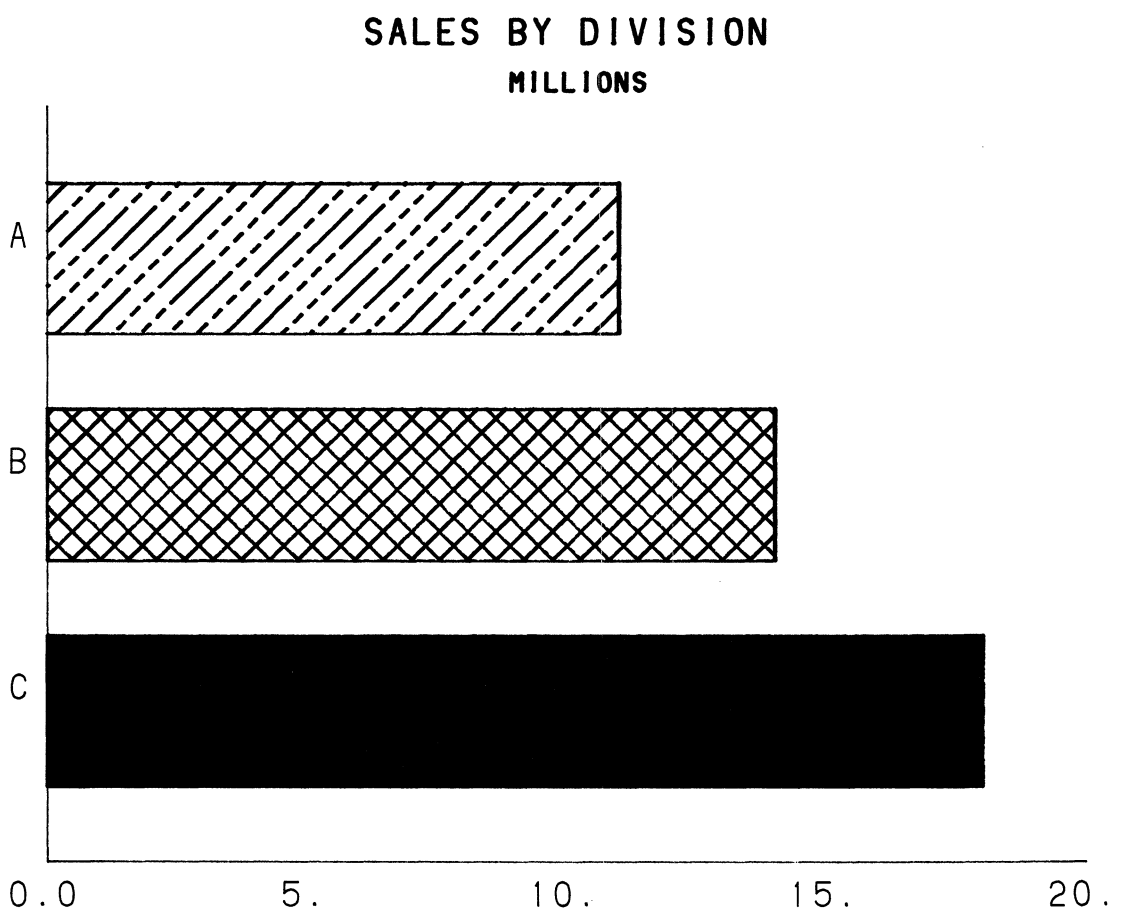
When you can't use color and line patterns are important, use the smallest or most dense pattern on the area of the chart closest to the axis and build up to the least dense pattern. (See Figures I & II.) When building pie charts begin by using the least dense line pattern for the biggest slice and end with the most dense pattern in the smallest slice. (See Figure V.) Be sure the pattern can reduce well. Also be sure all line patterns are at a 45 degree angle or this will produce an optical illusion. As always: Keep the pattern simple.

RULE 20 - ALWAYS REVIEW YOUR GRAPHICS!

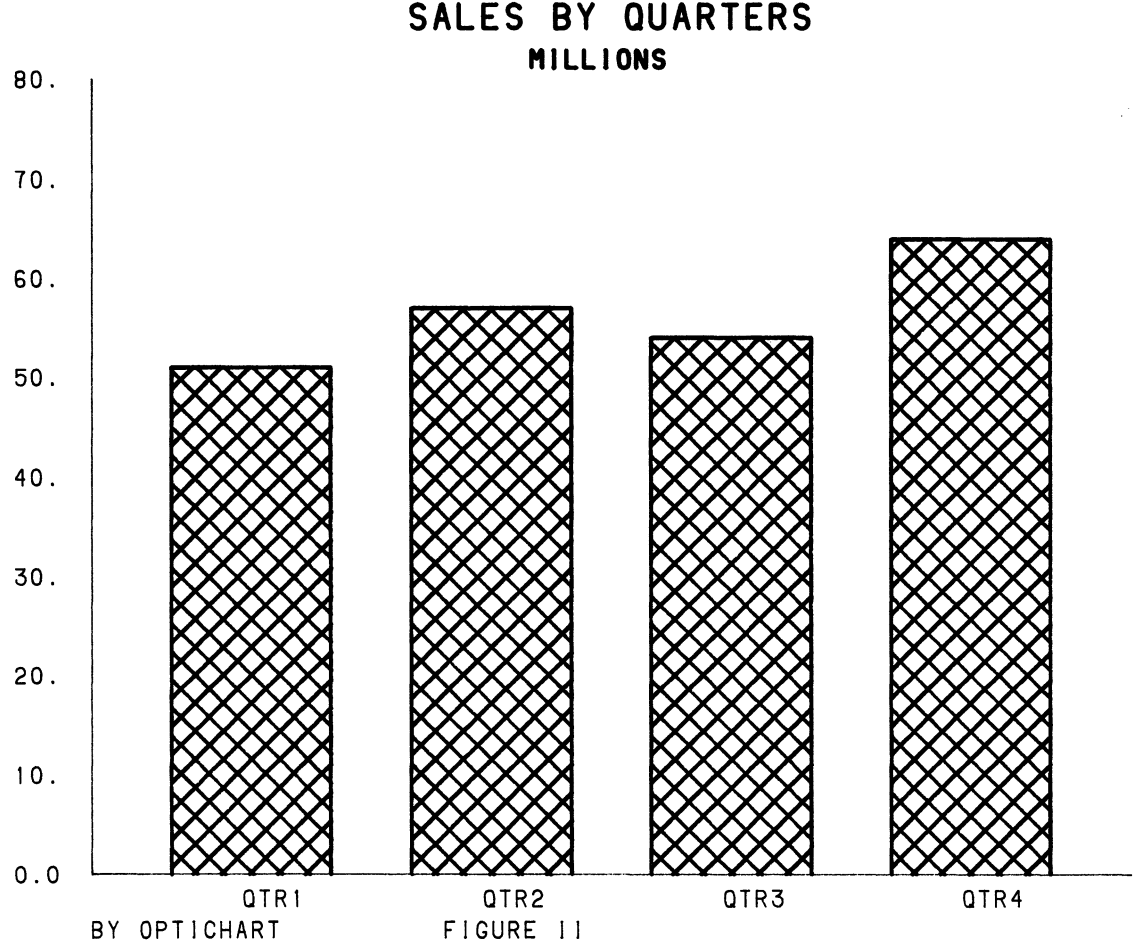
Ask yourself these three questions!
1) Has it been thoroughly proofed for errors and mistakes? (Always let two other people look at it.)
2) Does it convey my message without needing a verbal explanation?
3) Does it fit my audience?

If you can answer yes to all three, than you have a well prepared slide or overhead.

There! I have finished going over the Twenty Rules to Business Graphics. If you follow these rules, you should be able to reduce the time it takes to prepare your charts and graphics and increase their effectiveness and style.



BY OPTICHART FIGURE 1


SALES BY QUARTERS
MILLIONS
80.
70.
60.
50.
40.
30.
20.
10.
0.0
QTR1
QTR2
QTR3
QTR4
BY OPTICHART
FIGURE 11

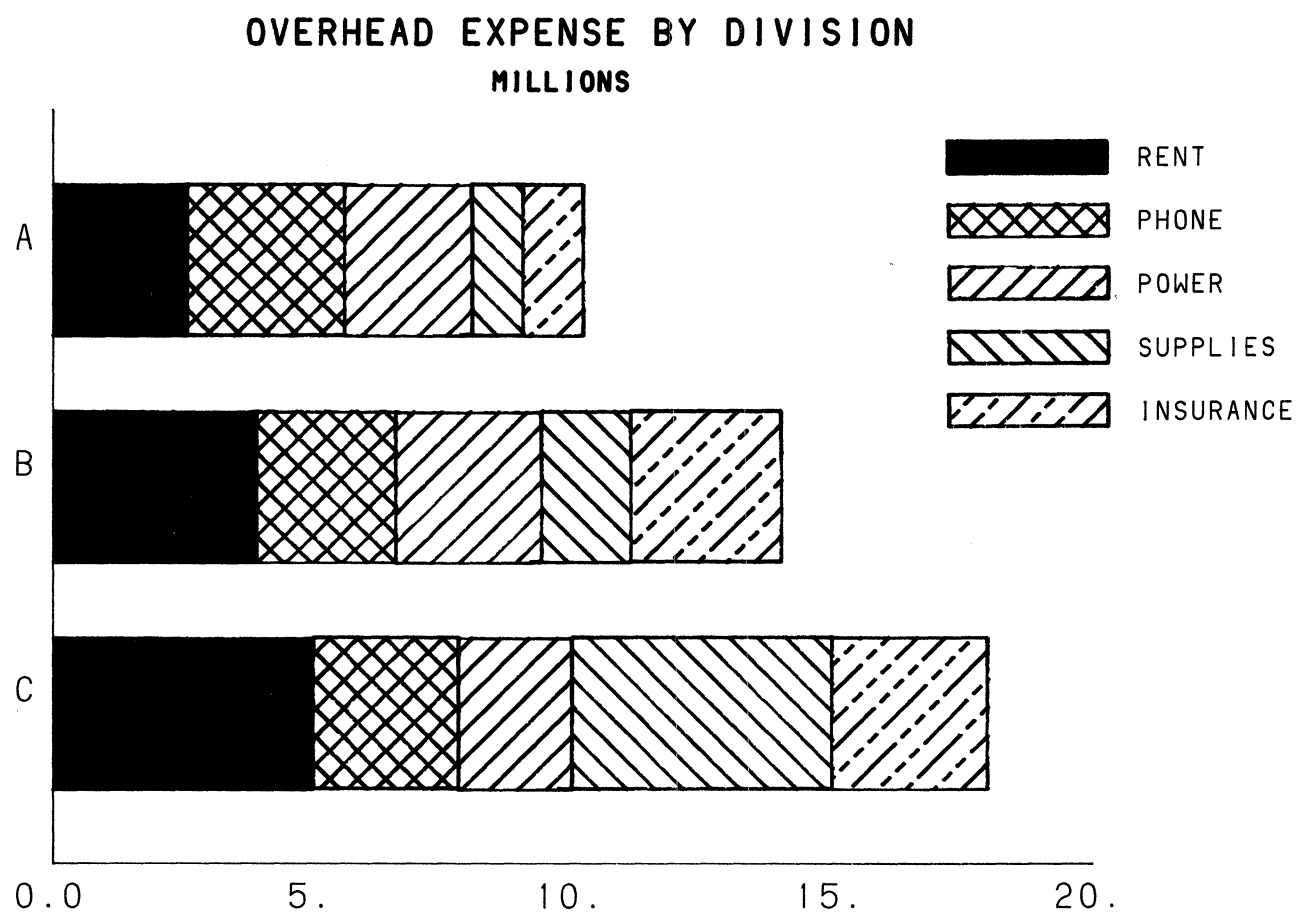
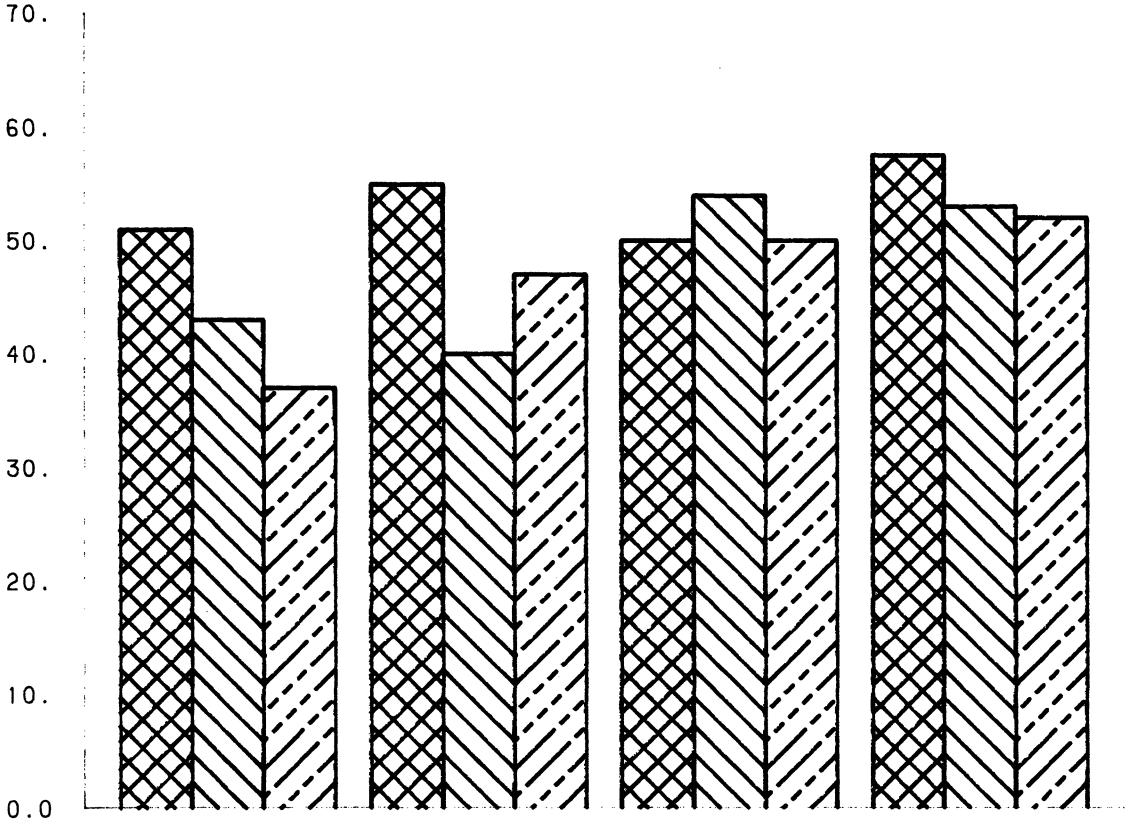

OVERHEAD EXPENSE BY DIVISION
MILLIONS
RENT
PHONE
POWER
SUPPLIES
INSURANCE
A
B
C
0.0
5.
10.
15.
20.
BY OPTICHART

FIGURE III

SALES BY DIVISION

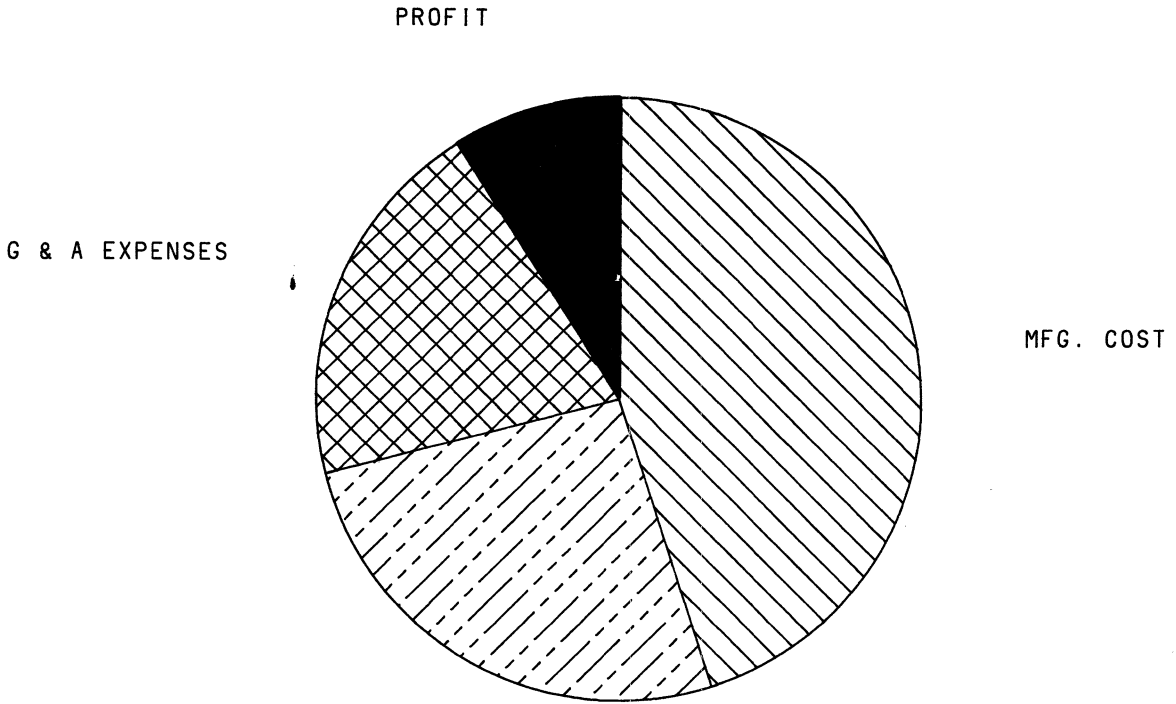
MILLIONS



BY OPTICHART

FIGURE IV

INCOME & EXPENSES
AS A PERCENT OF SALES

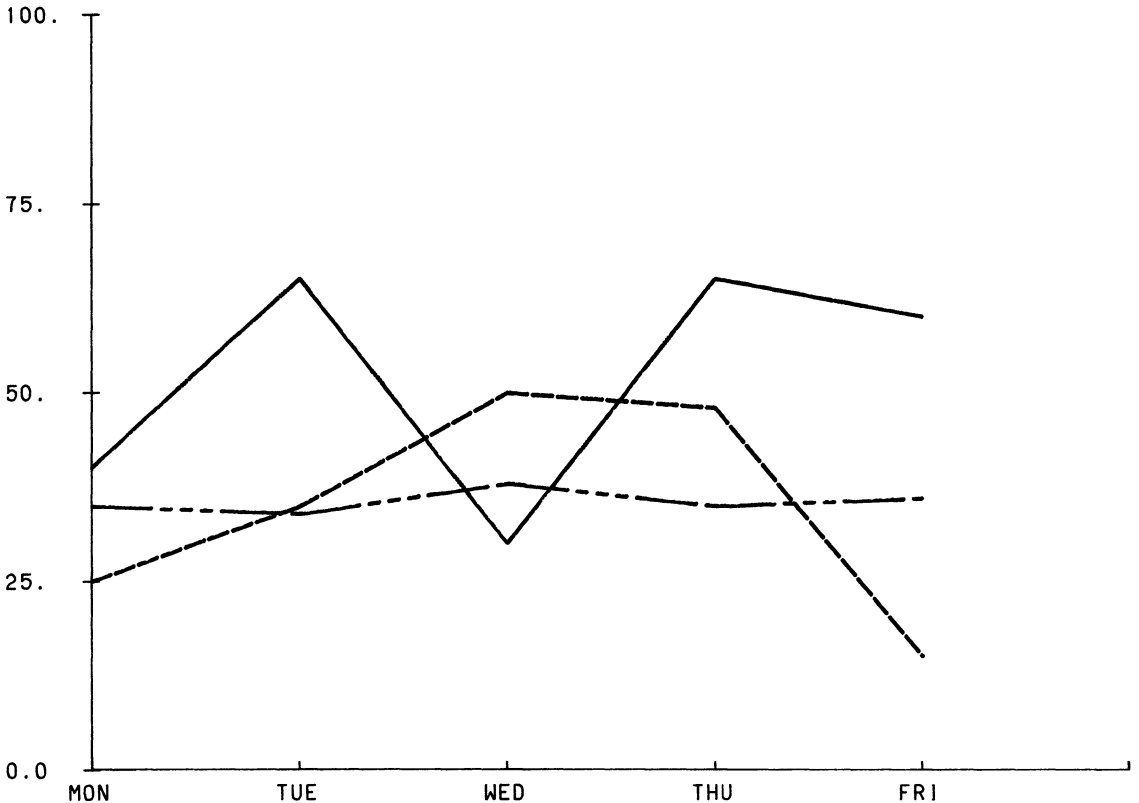


BY OPTICHART

FIGURE V

DAILY PRODUCTION BY PRODUCTS

THOUSANDS



BY OPTICHART FIGURE VI

MONTHLY SALES
MILLIONS

DIV.	A	B	C
JAN.	100	55	72
FEB.	112	63	73
MARCH	98	72	62
TOTAL	310	190	207

FIGURE VII

C THE FUTURE OF HP3000?

Mark W. Miller
JMA, Inc.
P. O. Box 270507
Houston, Texas 77277-0507

It all started back in the 1940's when ENIAC, the first modern computer, was built. A means of manipulating data (mostly numbers then) on the computer was needed. A crude instruction set was devised for the express purpose of processing numeric calculations. The process of programming was tedious, involving rows upon rows of binary data and instructions in the form of zeros and ones.

This was soon replaced with a higher level of instructions known as assembler. Assembler was a system of symbols representing the instructions which in turn were translated into binary machine instructions. This was the second generation of computer languages.

After some time came the third generation of computer languages which took programming a step further by using yet a more complex set of symbols to represent the instructions. Some of these symbols could be translated into hundreds or thousands of machine level instructions at compile time.

One of the first of these was a particularly good language for numeric calculations. It would take complex formulas and translate them into machine instructions. Scientists and engineers were elated. It became known as its acronym FORTRAN.

FORTRAN was widely used with computers to perform complex tasks that, until then, had been accomplished only with tedious manual calculations and involved countless manhours. A need was seen for more than just number manipulation and new third generation languages were sought. Thus, the proliferation of languages came forth for the scrutiny and acceptance of the computer programmers.

Some of these remained obscure; some became widely used; some flashed like a comet and died. Only a few have survived through the years with accepted standards: FORTRAN, COBOL and Basic are just a few. Newer and even more sophisticated languages were experimented with in the late 60's and early 70's. Out of this era came such languages as Pascal, APL, Algol and C. Here, we will focus on just one of these languages: C. We will discuss some of its features, advantages and disadvantages and include some tips on keeping your C code portable.

Why C?

Why would you use C when Modcal and SPL are the main systems languages on the HP3000? Why bother to learn a new language? There are several reasons.

".... C was created as a tool for working programmers. Thus, its chief goal is to be a useful language." (Note 1) And, indeed, it is friendly to the working programmer. It will do exactly what it is told to, even if it is improper or potentially self-destructive. C may not be the only language whose aim is to be useful, but most other languages have additional goals in mind. These separate goals have a tendency to get in the way.

C is modern in that it has control features that are desirable for modular design and structured programming. It incorporates many advanced features available in other languages and some that are not. Since it lends itself nicely to these other languages, it is easy to learn for those who have used such structured languages as Pascal or SPL.

C is efficient. I would venture to estimate that a C program can operate at about 90% of the efficiency of systems languages; possibly more if it takes full advantage of system features.

C is flexible. It lends itself as well to word processing as it does to systems programming or number-crunching applications. It is most unusual for a single language to be usable in such a wide variety of applications. The main advantage to this is that the development staff can get by with a single language for all occasions. Imagine all programs on your machine written in a single language. The ease of maintenance alone is justification for considering such a concept.

C can get down to bits and bytes. As a natural extension to the language, C allows use of assembly instructions to specifically manipulate the machine. These statements, although not portable, can gain speed and efficiency for the programs.

Probably the most outstanding reason to use C is that it can easily be made portable. Why re-invent the wheel? As time goes on, it is becoming more and more expensive to develop software. In order to cost justify large systems, portability can quickly become a savings factor. You can build applications on one machine and run them with little or no changes on another machine. More about that later.

---

1. Waite, Prata, and Martin, 1984, 13.

If these reasons aren't enough, consider the future. UNIX is quickly becoming one of the most popular operating systems on computers today. Most of UNIX was written in C and C is the preferred language under UNIX. It is estimated that 70% of all new personal computer software is being developed in C because of its powerful features.

C is one of the major languages supported by Hewlett-Packard on their Spectrum line of computers. Since SPL will not be supported by HP and Modcal is machine specific, C appears to be the language of choice for migration from the old HP3000 line to the new. Besides, most of the new HP Precision Architecture machines will also support a UNIX operating system (the series 840 already does), thus encouraging use of C. Further, the portability of the language makes it ideal for a multi-vendor shop.

Adding all these reasons together, they point to the distinct advantage of using C for development, system programming, or simple ad-hoc programs.

BRIEF HISTORY OF C

C began its life back in 1971 when Ken Thompson and Dennis Ritchie of AT&T's Bell Labs decided that some 'B' language features were a poor match for the PDP-11 minicomputers. Thompson and Ritchie had already developed the first working models of UNIX the previous year and had developed a word-processing system for Bell's Patent organization. Since the 'B' language had proven to be impractical, design of a new language began.

The design included features that could be efficiently modelled on most conventional computers. While intended for use primarily under UNIX, C has found its way to a wide range of hardware and operating systems. In 1973, UNIX was re-written in C and most subsequent versions retained this commonality with version 4 UNIX.

A tutorial and description of C was released in 1978 by Brian Kernighan and Ritchie named the "C Programming Language". This book became the definitive reference for the C language and is so popular, it is still in print and available today. This book coincided with the first informal release of UNIX by Bell Labs.

Through the years, UNIX and C have undergone changes and improvements too numerous to explain here. If you are interested in further details, see the references listed herein. The ANSI X3J11 committee released a formal issue of a Draft Proposed Standard for public comment which expired in March, 1987. This means that within a year or so, it will be released as the standard.

FEATURES, CONSTRUCTS AND PHILOSOPHIES

C comes to the HP3000 with a rich array of functions and programs already available. The lexical analyzer "lex" will parse token input quickly and efficiently. There is also a good syntax processing program known as yacc (yet another compiler compiler) that will analyze syntax. Many basic and rudimentary functions are built into C which allow taking advantage of the language as opposed to system specific functions. This indirect usage of resources makes the code more transportable from machine to machine. These functions and programs are available through many sources. Some of the compiler vendors offer them in library packages, available for an additional fee. Some of them are available through bulletin boards and some are available in print. Some of them can be ported from other computers.

While many of these functions are slower than system functions, the maintained portability can mean tremendous savings in terms of coding. Many of these functions are installed as macros (discussed later) which are translated to actual system calls. Decisions must be made to determine exactly how much should remain portable as opposed to system specific.

CONTROL FLOW

C has several constructs that would be of interest to Pascal and SPL programmers. One of these, the conditional operator, is a short-hand way of expressing a condition. It is a two-part operator with three operands. The general form of the conditional operator is:

```
expression A ?  expression B  :  expression C
```

It may be used anywhere that an expression is appropriate. If expression A is evaluated as true, the whole conditional expression takes on the value of expression B. However, if expression A is evaluated as false, the whole conditional expression takes on the value of expression C. This is handy when coding an unimportant if-then-else for simple assignments.

Example:

```
a = (b>0) ? b : -b;
```

The expression (b>0) is evaluated and, if true, assigns the value of b to a. If false, it assigns the value of -b to a.

In addition to these conditional methodologies, C has an else-if, multiple choice extension to the if-else structure. This allows a case-like structure without actually using the case structure. It operates like an exclusive if-then-else

except that there are multiple alternates instead of just one, only one or more of which will actually execute based on the conditions expressed.

Example:

```
if(a == 1)              /* most likely condition */
 {
  b = a;                /* assignment */
 }

else if(a == 2)         /* less likely condition */
 {
  c = a;
 }

else if(a == 3)         /* least likely condition */
 {
  d = a;
 }

else
 {
  e = a;                /* default assignment */
 }  /* end of else-if statement */
```

C also has a switch statement which is its' version of a case structure. Unlike the else-if construct, switch can allow one or more of the cases to be executed by "falling through" cases. The break statement is used to prevent this by transferring control to the first executable statement after the switch structure.

Example:

```
switch(a)               /* switch on the value of 'a' */
 {

  case 1:
         b = a;         /* value of 'a' is 1 */
         break;         /* break out of the switch */

  case 2:
         c = a;         /* less likely case */
         break;
```

```
        case 3:
                d = a;      /* least likely case */
                break;

        default:
                e = a;      /* default assignment */
                break;

        }  /* end of switch statement */
```

Another control flow construct in C is the infinitely abusable goto. By using labels (just like SPL and Pascal), flow may be re-routed to different sections of a function. This can make the program run faster, but it also makes the program source more difficult to understand. Sometimes, though, it is desirable when you need to bypass large sections of code or your code size is restricted by the computer you are using. This can be especially true of large applications on a PC. In any case, goto should be used sparingly to avoid radical code problems.

LOOPS

Looping constructs in C come in three basic flavors: for loop, while loop and the do while loop. The for loop consists of 3 components in its control expression as follows:

```
for(expression A; expression B; expression C)
```

Expression A is used for initialization. It may contain none or more expressions separated by commas. This section is performed only once at the beginning of execution of the for loop.

Expression B is the control of the loop. If present, it must be a boolean expression that evaluates as true or false. This expression will be tested before beginning execution of the loop on each iteration. If true, the loop executes; if false, control is passed to the first executable statement after the loop. This expression can be an empty expression (no boolean test), but if you use this method, be sure you have a way to break out or suffer an endless loop condition.

Expression C is a performing expression. It will be executed once at the beginning of the loop before the boolean test of expression B and before execution of the loop. An exception to this is the first iteration. It may also be forced to perform at the end of each iteration. It may be none or more expressions separated by commas.

The while loop has a condition which must be met for each iteration before the iteration. The do while loop tests the condition after each iteration before re-iterating. These expressions are rarely an empty expression.

Examples:

```
while loop
----------

while(a > b)          /* loop condition */
 {

  b = (b + 1);        /* loop action statement */

  }                   /* end of while loop */

do while loop
-------------

do                    /* begin loop */
 {

  b = (b + 1);        /* loop action statement */

  } while(a > b);     /* end of loop & conditional test */
```

STRUCTURES AND UNIONS

C provides a means for defining contiguous memory for multiple variable types. Similar to Pascal's record, C's structure allows assembling of "records" of data. This data may then be manipulated as a whole or in parts. It can be written to files, read into or even passed (as a pointer for most compilers) to other functions. A structure can even have multiple dimensions and be used to accumulate data like a COBOL table. It can also be self-referential by using a pointer of its own type. This is useful when constructing a B tree or a linked list.

Example:

```
struct date             /* structure tag declaration */
 {
  int day;              /* month day integer */
  int month;            /* month integer */
  int year;             /* year integer */
  int yearday;          /* day of year integer */
  char mon_name[4];     /* month name */
  };                    /* end of structure definition */
```

Memory can be re-used as in FORTRAN EQUIVALENCE or COBOL REDEFINES to facilitate efficient processing or translation of data. The union construct allows virtually limitless use of the same memory locations.

Example:

```
union utag              /* union tag declaration */
 {
  int ival;             /* integer value */
  float fval;           /* float (real) value */
  char *pval;           /* character pointer value */
  };                    /* end of union definition */
```

CASTING

One of the most powerful options of C is known as casting. By using casting, you can change the type of a variable or constant to use in expressions. Examples of this would be to cast a short or small integer to a long or double integer or to cast a short or small integer as a character or byte integer.

Example:

```
int i;                  /* integer declaration */
long j;                 /* long integer declaration */
char k;                 /* character declaration */

j = (long) i;           /* integer cast to long */
k = (char) i;           /* integer cast to character */
```

This is handy when storing data or passing data around from routine to routine. While it is a powerful feature, it can also cause many problems if a variable is cast improperly. This is especially true for pointers. It is easy to mis-cast pointers and cause corruption and/or bounds violations. Byte placement within a word can also be corrupted if care is not taken.

MACROS AND PRE-PROCESSOR COMMANDS

A major difference incorporated in C's philosophy that differs from most other language compilers is known as a pre-processor pass. During this pass, all macros and pre-processor commands are resolved.

Macros are a direct replacement for code fragments in a program.

Example:

```
#define MAXINT  32767
#define MININT -32768
```

Wherever you place MAXINT in your source code, the pre-processor will replace it with the value 32767. Wherever you place MININT, it will be replaced with the value -32768. This facility is especially useful for replacing constants with a macro name. Since these values happen to be the maximum and minimum values for a short integer on the HP3000, I will use them to cemonstrate the pre-processor commands feature. But first, let's look at another way of using the #define. It can be used to create a condition unique to its name.

Example:

```
#define HP3000
```

This allows the use of directives based on the fact that HP3000 is now a known (although not necessarily required) existing define condition.

Macros may also contain action statements to be performed with parameters. Many of the C functions are really just macros that replace your code at compile time.

Example:

```
#define SQUARE(a) (a*a)     /* the macro*/

int i = 2;
int j, k;

j = SQUARE(i);          /* j gets the square of i */
i = SQUARE(2);          /* i gets the square of 2 */
k = SQUARE(i);          /* k gets the square of i */
```

this fragment of code is compiled as:

```
j = (i*i);              /* j gets the square of i */
i = (2*2);              /* i gets the square of 2 */
k = (i*i);              /* k gets the square of i */
```

NOTE: The second expression is further refined by the compiler to resolve the constant calculations for the program to yield:

```
i = 4;                      /* the result of 2*2 */
```

Pre-processor commands direct the compiler pre-processor to compile according to their directives.

Example:

```
#ifdef HP3000                 /* if this is an HP3000 */
#define MAXINT 32727          /* use these values */
#define MININT -32768
#endif
#ifdef SPECTRUM               /* if SPECTRUM */
#define MAXINT 2147483647     /* use these values */
#define MININT -2147483648
#endif
```

By using these directives, you can control which set of values will be used by the compiler.

Another pre-processor command is the #include directive. The include directive takes two basic forms.

Examples:

```
#include <stdlib.h>
#include "stdio.h"
```

When the file name is enclosed in angle brackets (less than and greater than), the compiler will search the system directories for the file specified and process it just as though it were part of the source file. When the file name is enclosed in quotes, the compiler will search your working directory for the file first and if not found, will continue searching through the system directories. This command can be anywhere in your source, so it can be used to insert anything from variable declarations, to constants, and even sections of source code.

TYPEDEFS

Another unusual feature of C is typedefs. This feature allows the programmers to create their own or rename types of variables. This can be especially useful for novices who are used to another language.

Example:

```
typedef int INTEGER_2;
typedef long int INTEGER_4;
typedef float REAL_4;
typedef double REAL_8;
typedef char CHARACTER;
```

For a FORTRAN programmer, this would make recognizing variable types simple and straight forward.

Another method of using typedefs involves creating names for commonly used types.

Example:

```
typedef int boolean; /* 16 bits used as a boolean */
typedef char flag;   /* 8 bits used as a boolean */
```

The typedef can be used to create commonality in programs by recognizing the regularly used types from a different standpoint.

LONG JUMPS AND SIGNALS

A very special set of constructs for control flow are available in C. C programs are usually a collection of many small (easily maintainable) functions. To accomplish various tasks, usually one function calls several functions, which in turn calls several functions, etc.

Error and exception handling in this situation becomes a problem unless you wish to pass error and/or exception flags between all functions. To circumvent this problem, C has two special functions (setjmp and longjmp) and a special typedef (jmp_buf). In order to understand these, first we must explain the concept.

When a function is called from another function, the current status and information about the calling function must be preserved for when the called function returns. This is stored in what is known as a stack frame. The volatile variables, stack pointers and some registers are stored in this stack frame for later recall. As each new subordinate level is reached, another stack frame has been added. These pile up as deep as the nesting and recursiveness of functions in your program. As the functions return and "unnest", these stack frames are removed and the stored stack frame in jmp_buf is used to restore the environment of the calling function for continued processing.

EXAMPLE: Suppose you have a program with three functions: first, second, and third. Further suppose that first calls second and second calls third as in:

```
first > second > third
```

NOW THE QUESTION ARISES: What if an error or exception occurs in the third function and control needs to return to the first function?

ANSWER: By using the setjmp function in the first function, we can, in effect, store a special copy of the stack frame in a type jmp_buf declared variable.

Then, in the third function, we can call the longjmp function with the copy of the stack frame when an error or exception occurs. The longjmp function will then "unwind" the stack by discarding the stack frames "after" the first function and returning control to the place in the first function where setjmp was originally called.

Note that control cannot go the other way (downward nesting order), but the same stored environment may be shared by many different subordinate longjmp function calls. Also, note that an additional integer parameter is part of the longjmp function call, allowing you to return different values and thus handle many different possibilities.

On many UNIX and MS-DOS systems, the hardware and operating systems software provide a means of communicating exceptions to programs. Exceptions are events external to the user program which will affect the program's process. This event may be anything from floating point overflow to a power fail to a control-y interrupt.

In order to deal with these exceptions, C has a special function known as signal. By calling the signal function, you can identify a function which will automatically be executed if the specified signal is received by the program from the system software or hardware.

By specifying this function, the program can optionally attempt to recover (if recovery is feasible), or gracefully shut down (closing files, etc.) depending on the signal received. The signal function, although very powerful, is very machine specific in that different machines have different signal meanings.

ADVANTAGES AND DISADVANTAGES: TIPS AND TRICKS

POINTERS

One of C's features is the use of pointers and, of course, all of the "gotchas" that are inherent with their usage.

Because C allows usage of pointers and because the compiler has no knowledge of where these point to during run time, all written code is unforgiving. Of course, on the HP3000, MPE will keep you "in-bounds", but nothing will keep you out of your own data area or prevent you from "walking on" your other data. Sometimes a ghastly error in logic can cause great grief in locating program bugs.

If you are fortunate enough to have a symbolic debugger, this type of error can be traced down to its' perpetrator and eliminated. One technique I found in accomplishing this is what I call "locate by timing". Once the data that is being corrupted is located, start the program over again and

periodically check the "to be corrupted" data variables, noting when the corruption takes place. Repeat this process and narrow this down through a bisecting technique until your culprit is found. It can be any action statement, but I found most of mine to be simple errors of omission or a typo.

The indirection operator (*) is used to indicate "contents of" for a pointer. It is easily abused when performing pointer arithmetic.

Example:

```
*string + i = 'A';      /* wrong */
```

is not the same as

```
*(string + i) = 'A';    /* correct */
```

The first represents an error and will either give unpredictable results or will yield a compile error. The second will load the letter 'A' into the contents of the address of (string + i).

The means of loading a pointer are different between arrays and non-array variables. Since arrays are actually an address pointer, loading is expressed as:

```
pointer = array;
```

whereas an integer would be expressed as:

```
pointer = &integer;
```

The address operator (&) is used to load a non-array variable address or the specific address of an element within an array into a pointer.

PASSING POINTERS

When passing addresses through to lower routines, be sure you are passing an address and not an address of an address. The simplest method of passing an address through to a lower routine is to merely reference it without any changes whatsoever. This will ensure that the address is passed without corruption. Another method of avoiding problems is to copy a value into a local variable and pass its' address instead. If you do this, be sure to pass the value back to the calling routine (assuming it changed) before leaving the routine.

Note that this is expensive in terms of speed of program operation. Sufficient usage of this method could cause a significant degradation of the program. Copying the value to a local variable costs in time and the extra memory required to hold this data costs in time indirectly. How? Beside the obvious memory space, consider that each time the routine is invoked, its memory must be allocated, sliced into areas for the variables and addresses assigned and noted. Each time the routine invokes another routine, it must eventually build a stack frame to hold these local variables and their values in for continued use upon return.

Extra diligence must be given especially with pointers to pointers. A pointer to a pointer is useful for passing through to functions or for abstracting data to give the "Black Box" effect with a flexible function. These are powerful constructs; but easily abused. The key to successful usage is to be sure it is initialized properly with the address to the desired pointer each time prior to use. This prevents loading a previous pointer with the intended current pointer value.

STRINGS

One of the aspects of C that stands out from other languages is the way strings are handled. As with SPL and Pascal, a string is stored in a byte array, but that is where the similarity stops. C distinguishes between a byte array and a string by ending a string with a null character (Ascii zero). This is the main means of determining whether a byte array contains a string or not and determines the actual end of string. Such basic functions as determining a string's length, upshifting alpha characters or simple movement in memory require a null terminated string.

In order to deal with this anomaly, I suggest putting a null terminator on every string that is brought into your program immediately. This may require that your byte array buffers be declared with one extra byte each. If so, you always have the option of initializing that last byte to null. As a result, some memory is wasted, but process time would be reduced. Furthermore, a simple routine to strip trailing blanks can remove any excess blanks by padding the end of the byte array with nulls. This preserves the trailing null, but don't forget to leave it there when re-initializing the byte array.

STRING COMPARISONS

In most other languages, string comparisons are somewhat a straight forward affair. In C, there are some peculiarities inherent with the built-in string comparison functions. As

mentioned above, many string related functions require a null terminated string. This is also true of the strcmp (string compare) function. It can, however, tell you that two strings are equal even when they aren't.

Example:

NOTE: \0 is a null character.

```
#define STRLEN 20

char stringa[STRLEN];  /* contains "ABCDEF\0" */
char stringb[STRLEN];  /* contains "ABCDEFGH\0" */
int ch;                /* integer declaration */

ch=strcmp(stringa,stringb);
if(ch == 0)            /* if strings are equal */
 {...
```

In this example, ch would be zero because every byte in stringb that corresponds to each byte in stringa are equal up to (but not including) the null terminator in stringa. The alternate function strncmp (finite byte array comparison) could be used if one of the strings are not null terminated.

Example:

```
#define STRLEN 20

char stringa[STRLEN];  /* contains "ABCDEF\0" */
char stringb[STRLEN];  /* contains "ABCDEF " */
int ch;                /* integer declaration */

if((ch=strncmp(stringa,stringb,STRLEN)) == 0);
 {...
```

NOTE: The if has been consolidated with the strncmp function expression here to demonstrate the compactibility of C code.

In this example, ch would not be zero because the null character in position 7 of stringa is not equal to the blank in position 7 of stringb.

All of this confusion may be overcome by following some simple rules:

1. Null terminate all strings.

2. Compare lengths of strings as well as comparing values to ensure true equality.

This second rule becomes truly important when your program manipulates numerous similar strings. You may opt to write your own string comparison function or macro and not even use the supplied functions. Remember, though, that the macro is the preferred method because it will always execute faster than calling a function. In either case, follow these rules for successful comparisons.

I/O

"Input and output facilities are not part of the C language..." (Note 2). Since input and output were not designed into C, and in the real world, all programs use I/O, a "standard I/O library" was developed for C. Most C compilers come with this rich library of routines to perform I/O on a standardized basis.

Samples:

```
putchar()   /* output a single character */
getchar()   /* input a single character */
printf()    /* formatted output */
scanf()     /* formatted input */
fwrite()    /* binary, fixed-length record file write */
fread()     /* binary, fixed-length record file read */
```

C is not oriented towards fixed length records, but instead towards a "byte stream." A "byte stream" is like a stream of data coming into or flowing out of a program. It is delimited by such things as newlines, nulls, tabs and, of course, end of file indicators (which, along with the blank character are collectively known as white space characters). There are no end-of-record indicators other than newline characters. Since these newline characters may appear anywhere in the stream, the records are essentially variable in size. Note that the end of file and newline characters are also machine dependent.

This is common on UNIX machines; however, it is adverse in relation to the fixed record blocking efficiency of most non-UNIX minis and mainframes. Regardless of this factor, the C library does have I/O access methods (fread and fwrite) for dealing with fixed-length binary records which lend themselves nicely to the fixed-length format predominant on the HP3000.

---

2. Kernighan and Ritchie, 1978, 143.

FUNCTION ARGUMENTS

In C, a program usually consists of many functions. There are essentially three types of functions in C. The first of these, the main function, occurs exactly once in each program. It is the programmer's outer block for the program, and it is the first user-written function to begin execution when the program is run (C does not currently support auxiliary entry points).

The second kind of function is the typed function. This kind may occur as many times as needed in a program. It is assigned a specific type when declared, and will return a value of that type every time it is called in the program. This value may or may not be used by the program to make decisions or complete computations or an expression.

The third kind of function is the void function. When a function is declared type void, it does not return any value when called (except through arguments, if any). The void function is analogous to FORTRAN subroutines or Pascal procedures.

A potential problem area in C programming is in passing parameters to functions. C does not automatically type match parameters in function calls. This means that if you pass the address of a short integer in an argument that is looking for the address of a long integer, the called function may use the addressed area as though it is a long integer; thereby corrupting memory that happens to follow the short integer. Another error would occur if the function was expecting an array of integers.

References in the called function may indicate array subscripts or use pointer arithmetic which again would corrupt memory if passed a simple integer address. To avoid this, either use the compiler argument type checking (if your compiler supports it); use the syntax checking utility lint (if you have access to it); or keep hard copies of your functions handy and check the arguments carefully when coding.

ASSIGNMENTS

Another fundamental difference between C, Pascal and SPL are the assignment and comparison operators. The assignment operator in C creates the most disturbing problems I have personally encountered. The C assignment operator (=) is easier to type than the comparison operator (==), thus in condition expressions errors may easily occur.

Example:

```
if(today = 320)
 {...
```

In this condition expression, the condition will always be true because the value 320 will be assigned to today and since the value is non-zero, it is true. (In C, 0 is false and non-zero is true.) The philosophy for this was that more assignments are used in most code than comparisons, thus less typing is required. The extra colon required in Pascal and SPL to assign values makes you think about your expression. The extra (=) in a C comparison is easy to forget and forgetting it can cause many problems in debugging a program.

INITIALIZATION

Another weakness of C is in compiler initialization of arrays, unions and structures. Unlike Pascal and SPL, all initialization values must be explicitly provided in order for the compiler to initialize them. This is most inconvenient when initializing a large, multi-dimensional array. Typos can easily occur and cause problems. I suggest double-checking the values for accuracy, presence (as opposed to unintentional absence), punctuation and order (since all arrays are really linear).

STANDARDIZED ENVIRONMENTS AND MAINTAINING PORTABILITY

I've mentioned several times about the portability of C. Actually, portability is a relative term. Some would consider portable to mean that a program runs properly as soon as it is re-compiled on the new system. Others may consider it portable if it runs after some modifications. In reality, a program that performs any significant tasks will most likely need adjustments when ported to a different computer.

No computer language is automatically portable. However, by taking proper precautions the code can be developed with portability in mind and make the task simpler. There is one major key to building portable code: Isolate machine idiosyncracies.

This is no easy task, but given the fact that if all machine dependencies are isolated, the rest of the code is automatically portable! Then it becomes a matter of implementing workable alternatives for the isolated code sections.

Our purpose here is not to point out the specifics (books can be filled with them), but instead to provide some general guidelines that will help to diminish the task.

1. KNOW THE MACHINE YOU ARE PORTING TO.

Know what computer or computers you will be porting the program to. Obtain manuals, books, notes and any materials that will help you to fully understand the features of the target machine. Select features that are necessary for your program to operate efficiently and productively. Note any special requirements these features have. Remember, the more specific features you use on any one machine makes the code that much less portable.

2. SELECT COMMON FEATURES

Determine what features are common to all targeted machines and use these features to your advantage. Refrain from using machine specific features as much as possible. Separate any specific features by using include files and keeping dependent features in specific routines. Avoid using system calls and keep interfaces as abstract as possible.

3. ISOLATE MACHINE DEPENDENCIES

Utilize C features that aid in keeping code transportable.

Examples:

#define - use defines to isolate machine specific constants which are used in data manipulation. Also good for setting a condition for each different type of machine.

#ifdef - useful for compiling separate sections of code based on the defined type of machine (see above).

#include - used to include machine specific code or machine specific declarations.

These pre-processor commands can aid in isolating machine dependencies and treat code according to the type of machine it is being compiled on.

Other areas to look for differences include: system and file access; file search patterns, naming conventions, characteristics and access paths; integer and float sizes, representations, alignment and formats, word boundaries, character alignment, bit order and most and least significant bytes; order of expression evaluation, and preservation of meaningfulness in expression evaluation; character sets, comparison and sort order; variable names (length limitations) and initialization.

4. AVOID FANCY MANIPULATIONS

Avoid bit manipulation, byte alignment, data transformations and other fancy footwork unless it has been abstracted to deal with the different machines.

5. STANDARDIZE LIBRARIES

Standardize your libraries to isolate machine specific properties and stick with ANSI standards. Avoid special features of different compilers.

6. TEST YOUR ASSUMPTIONS

When in doubt, write test programs to test your assumptions. Do this during the design stage to avoid ill design from the beginning.

7. USE MACROS FOR CONSTANTS

Avoid "hard-coding" constants. These should be taken care of with the macro facility of C to give them names. It is always easier to change a macro definition than to search source code for every usage. Additionally, the code becomes more readable with the macro text in it than constant values.

8. DEVELOP STYLE AND STANDARD RULES

Develop a coding style and standard. Stick with this style and make changes to it only when demonstrated to be beneficial to all affected systems.

9. DOCUMENT YOUR DISCOVERIES

Document differences, coding workarounds, idiosynracies and quirks as they are encountered. A properly maintained set of notes is extremely useful in a development environment. By sharing this knowledge, fewer errors will be propagated. This will aid in any future development or ongoing development.

Obviously, this is not a complete set of rules to follow for portability, but merely a guiding line to start with. Different shops use different methods and these rules and guidelines are suggestions from which a sensible standard may be developed and set. Further study of machine differences and commonalities will aid in this endeavor.

The ultimate goal is to glean a standard by which all newly developed programs could be ported to other computers quickly and efficiently. Further, programmers will begin to understand the common base by which all programs may be written for portability.

CONCLUSION

We've covered many of the features, constructs, advantages and disadvantages of the C programming language. There are numerous aspects that have not been mentioned. After all, this is not intended to be a tutorial, but merely an introduction. Perhaps your interest is piqued.

While C is certainly not the only language available, I consider it to be one of the better ones because of its power, speed, flexibility and portability. The future of software is gradually departing from single vendor machine dependence: a trend we dare not ignore. We may get by with "that old COBOL system" for a few more years, but not much longer.

Integration of personal computers and the advent of LANs, switching networks and other sophisticated communications resources are rapidly expanding the old definition of "data processing" and turning it into "information processing". The concepts are exploding into multi-machine interconnection, communications and cooperative processing. These new concepts will need new techniques to deal with the machine differences. One of these is to write portable software.

You may say that your shop only has an HP3000, but that does not eliminate the possibility of other brands or an upgrade to a Spectrum. After all, not all companies can get by with just one type of computer forever.

REFERENCES

Al Stevens, "C Development Tools for the IBM PC" (New York: Prentice Hall Press, 1986).

Brian W. Kernighan and Dennis M. Ritchie, "The C Programming Language" (Englewood Cliffs: Prentice-Hall, Inc., 1978)

J. E. Lapin, "Portable C and UNIX System Programming" (Englewood Cliffs: Prentice-Hall, Inc., 1987).

Kenneth Pugh, "C Language for Programmers" (Glenview: Scott, Foresman and Company, 1985).

Mitchell Waite, Stephen Prata, and Donald Martin, "C Primer Plus", (Indianapolis: Howard W. Sams and Company, 1984)

Narain Gehani, "ADVANCED C: Food for the Educated Palate", (Rockville: Computer Science Press, 1985).

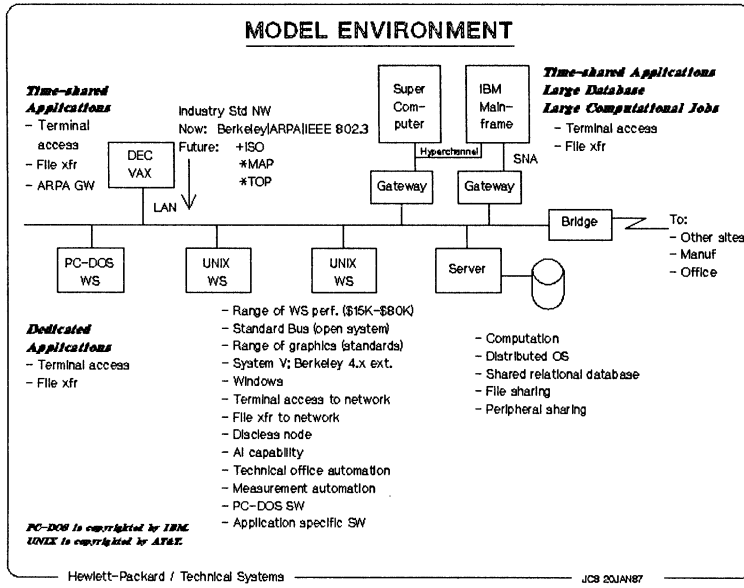
HP AdvanceNet for Engineering

Dave Morse

Hewlett-Packard Company
3404 East Harmony Road
Fort Collins, CO 80525

INTRODUCTION

As one of the five solutions in the HP AdvanceNet offering, HP AdvanceNet for Engineering addresses the networking needs of technical professionals engaged in engineering and other technical pursuits. This solution features the same emphasis on standards common to the other solutions. The solution is best understood by considering a model computing environment for engineering.



The diagram of the environment shows many of the key characteristics of both the computers and the network. A major trend in the engineering area in the past few years has

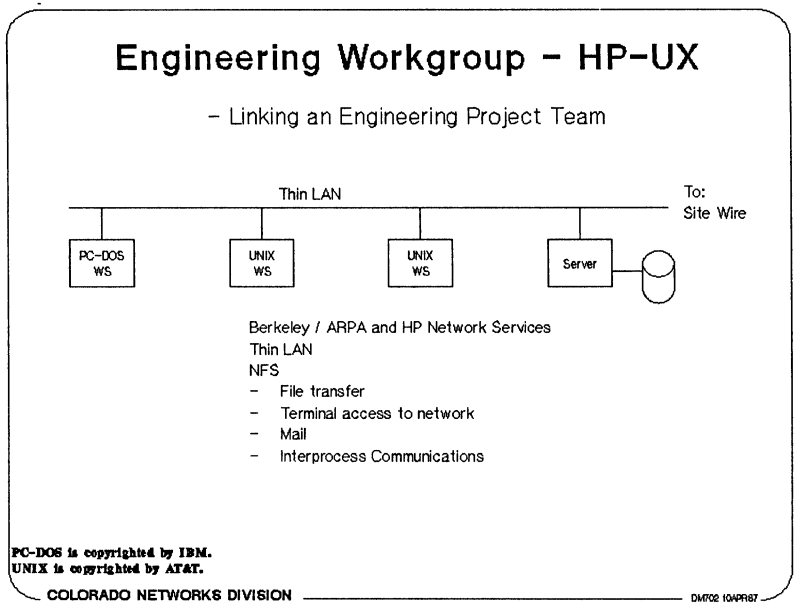
been a move to engineering workstations and acceptance of the UNIX operating system as a defacto standard. These workstations offer many advantages in terms of powerful graphics and consistent performance; but in order to be effective, they must easily integrate with the installed base of timeshare computers and other larger computers which may be added in the future. The resulting environment represents a range of computing power from personal computers to mainframes and super computers. In almost all cases, these computers will be supplied by several different vendors. In order for users to realize the maximum benefit of this environment, they should retain the desirable characteristics of the timeshare environment - easy information sharing and centralized system management - and also gain the benefits of the workstations in terms of distributed computing power. The network plays the key role in providing this.

The basic purpose of the network is to provide information and resource sharing. Users should be able to transfer files from one computer to another, log on to other computers, run applications on other computers, run applications on a local computer using data on remote computers, access peripherals connected to any computer and, in general, make the best use of the available resources to perform a wide variety of different tasks. In fact, it is not possible to do an effective job of providing computing for engineers without providing the supporting network.

The Engineering Solution, like the other HP AdvanceNet solutions, is comprised of modules. There are five modules in the Engineering Solution: Engineering Workgroup, Engineering Computer Center, Site Computer Center Access, Site Wire, and Company-wide Access. Each module consists of a collection of products which together meet the user requirements.

The first three modules represent a three-tiered hierarchy commonly found in engineering environments - workstations, super minicomputers, and mainframes. The workstations and super-minicomputers are often administered by the engineering department. The mainframes are often facility resources administered by the EDP or MIS departments.

ENGINEERING WORKGROUP MODULE



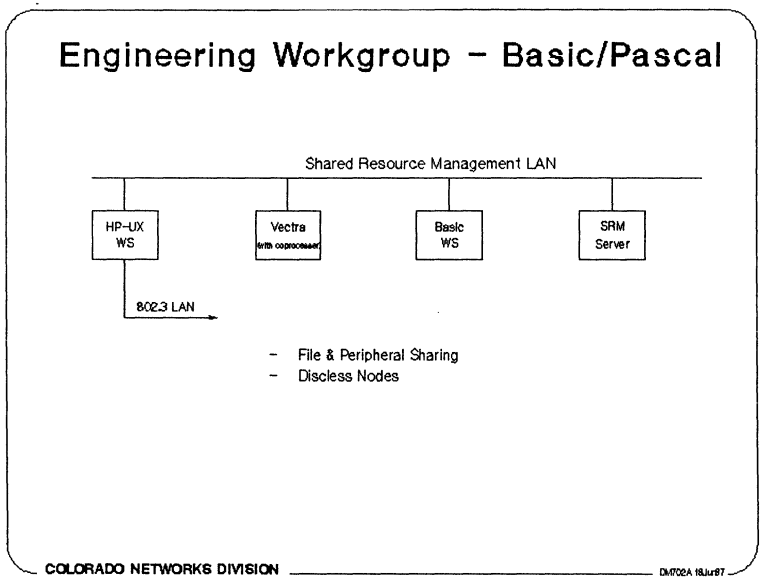
The most effective and productive way to connect a group of workstations is with a local area network (LAN). The LAN that has emerged as a standard for engineering networks is the IEEE 802.3 LAN. Early engineering networks utilized the Ethernet protocol, which served as basis for the IEEE 802.3 standard. HP offers IEEE 802.3 LAN as the basis for the engineering workgroup of UNIX workstations. For compatibility with existing networks, the Ethernet protocol is also supported. The IEEE 802.3 standard only defines part of the protocols necessary to provide communication among the computers. The other protocols employed are the Transmission Control Protocol (TCP) and Internet Protocol (IP) and the Berkeley and ARPA network services. The IEEE 802.3 standard defines the link used to connect the computers, TCP/IP provides a reliable connection from one computer to another, and the Berkeley and ARPA network services provide the specific functions required, such as file transfer, virtual terminal, etc.

The IEEE 802.3 standard allows for two types of cable - thin and thick. Because of ease of installation and

configuration, HP recommends use of the thin cable for the engineering workgroup.

One necessary capability not provided by either Berkeley or ARPA services is the ability to share files without copying the entire file from one computer to another. HP has augmented the Berkeley and ARPA services with an HP developed service called Remote File Access. Recently, a service known as the Network File System* (NFS) has been endorsed by a number of vendors and has emerged as a defacto standard for file sharing. One of the major advantages of NFS is that it is independent of the operating system and thus allows sharing of files among computers with UNIX and other operating systems. HP has announced NFS for the HP 9000 computers with initial shipments planned for late 1987.

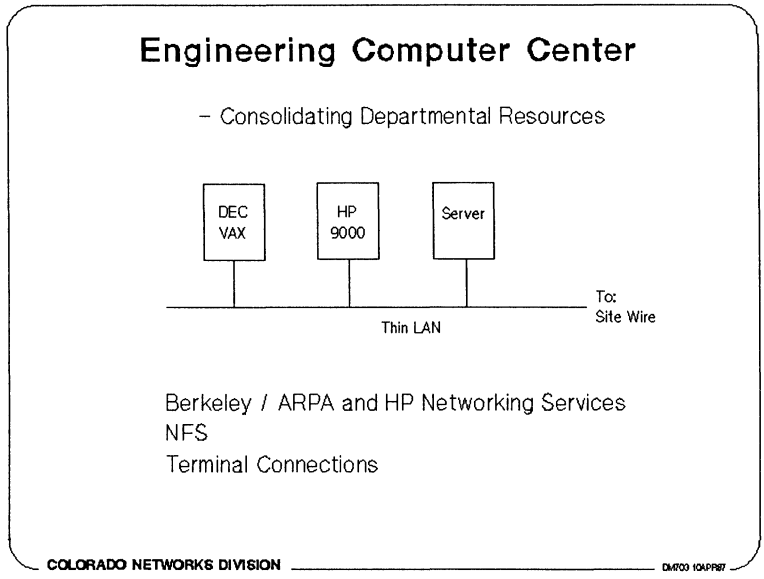
* NFS is a a trademark of SUN MicroSystems



Many engineering applications require use of computers to control various types of test and measurement equipment. HP offers several computers optimized for this task. One widely used computer is an HP 9000 Series 200 or 300 running the BASIC operating system (Rocky Mountain BASIC). The HP network for these systems is the Shared Resource Manager (SRM). The SRM features peripheral and file sharing and allows operation of the workstations without local discs.

SRM also supports the PASCAL workstation and HP-UX. SRM networks can be connected to IEEE 802.3 LANs through a workstation running HP-UX or a Vectra PC with the BASIC co-processor.

ENGINEERING COMPUTER CENTER MODULE



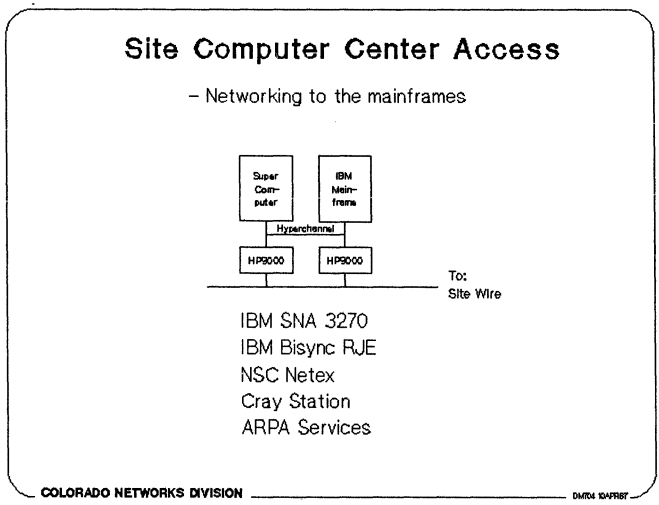
The engineering computer center represents the second tier in the engineering computing hierarchy. Computers in the engineering computer center are typically departmental resources, shared by several project teams. A timeshare super-minicomputer, such as the Digital Equipment Corporation VAX is very commonly used in this environment. Recently introduced HP Precision Architecture computers such and the HP 9000 Models 825, 840, and 850 will be installed here.

The engineering computer center could also house various types of servers for the engineering workgroups. These servers could manage large ensembles of discs or other peripherals such as laser printers. An advantage of putting servers in the engineering computer center is that they are centralized with the timeshare computers for convenient disc backups. Placing the majority of the discs and other peripherals here also isolates the workgroups from the noise generated by these devices.

HP's recommended wiring for the engineering computer center is again the ThinLan cabling. This allows for easy configuration of the computer center and permits convenient reconfiguration as necessary. The ThinLan network for the computer center can then be connected to the site backbone for communication with other workgroups or computer centers.

The ARPA and Berkeley network services can be used with any computers running UNIX. For example, the HP 9000 Series 800 computers can all be configured with ARPA and Berkeley network services to augment the HP-UX operating system. DEC VAX computers running the VMS operating system can be equipped with ARPA services via software packages available from DEC and several third parties. An alternative means to connect DEC VAX computers with VMS is to install HP Network Services on the VAX computer. This product provides HP AdvanceNet Network File Transfer (NFT), allowing file transfers between the VAX and any HP computer supporting NFT. The HP Network Services for the VAX run in user space and utilize standard DEC LAN hardware, permitting coexistence with DECNet. The engineering computer center would also provide terminal connections for the various timeshare computers.

SITE COMPUTER ACCESS MODULE



The site computer center is the province of the mainframe and supercomputer. IBM and IBM compatible mainframes are commonly found here. Engineers utilize these resources to

execute jobs requiring extensive computational power or to access large databases. Because of the dominance of IBM in this environment, required networking capabilities fall into two categories - IBM communications and "other".

Today the most commonly used protocol to communicate with IBM mainframes is IBM Systems Network Architecture (SNA). An older protocol, Binary Synchronous Communications (Bisync), is still in use in some installations. Either of these alternatives offers convenient communications to IBM because the engineering computers emulate standard IBM devices, such as interactive terminals or remote job entry stations. From the IBM mainframe's perspective, it is communicating with another IBM device. This greatly simplifies the task of the mainframe system managers, since they deal with standard IBM software. HP offers both SNA and Bisync communications products for communication with IBM mainframes.

A disadvantage of utilizing standard IBM SNA communications is that the performance is generally limited to that attainable over 56 Kbit/second links, far short of what can be obtained with a LAN. Because of the performance limitations, many site computer centers support alternative, non-IBM, connections to the mainframes.

Probably the most commonly encountered product is Hyperchannel, provided by Network Systems Corporation (NSC). Hyperchannel features a 50 Mbit/second link and supports a wide variety of computers in addition to IBM. Hyperchannel connections are available from NSC for HP 9000 computers.

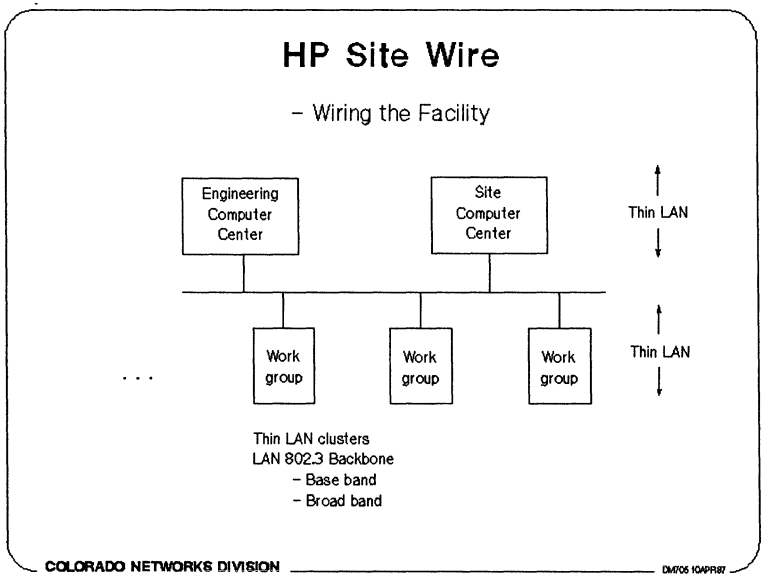
It is also possible to support the ARPA services on an IBM mainframe. Products are available from a variety of vendors. IBM also sells a TCP/IP/Ethernet product. Many of these products are new on the market and are not commonly installed. Where they are supported by the site computer center, they offer an additional high speed connection from the HP computers to IBM.

Many site computer centers also contain supercomputers such as Crays. Cray computers running the Cray Operating System (COS) support access via a protocol called Cray Station, which runs over 50 MBit/second Hyperchannel hardware. Cray Station software is available from Cray Research for the HP 9000 computers. Cray computers running the Cray version of UNIX (UNICOS) support ARPA services over an Ethernet LAN and can communicate with HP 9000 computers using this protocol.

In general, communication with the site computer center will

be via gateways between the engineering or facility LAN and the computer center. The gateways provide access to the computer center for other computers on the network and eliminate the need to install individual mainframe communication links for each computer. HP 9000 Series 300 computers can serve as these gateways.

HP SITEWIRE MODULE



The network of choice for most engineering applications today is IEEE 802.3. IEEE 802.3 supports two cabling options ThinLAN and ThickLAN. HP's recommended wiring scheme utilizes ThinLAN clusters for the engineering workgroups and engineering computer center. These ThinLAN clusters are connected to a ThickLAN backbone which runs throughout the facility. A device known as a ThinLAN Hub provides the connection between up to 4 ThinLAN subnets and the ThickLAN backbone. The ThinLAN and ThickLAN segments run at the same 10 Mbit/second link speed.

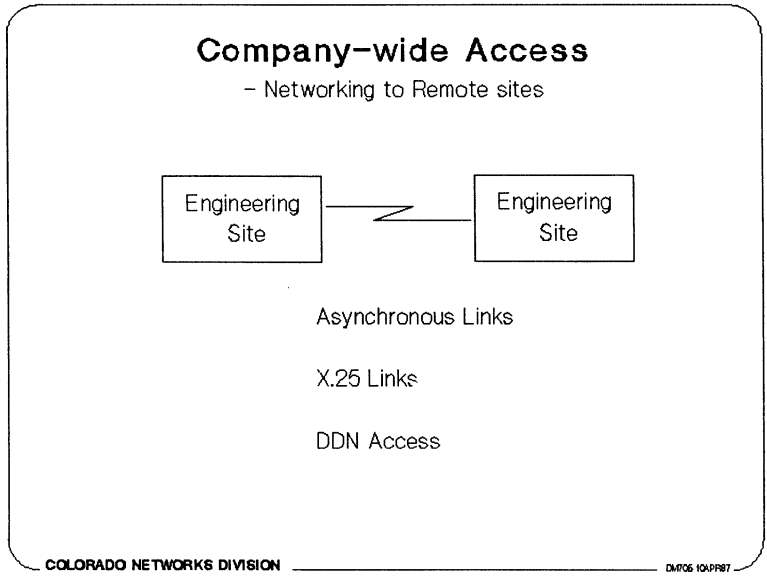
Small engineering networks can be created by using only a single ThinLAN network or by connecting up to four ThinLAN clusters with a single ThinLAN Hub.

ThickLANs can serve as backbones for networks of 1-2 kilometers in length. ThickLANs are baseband networks.

Broadband backbones are utilized to connect larger facilities or campuses. Broadband backbones use cable television technology to cover distances spanning many kilometers. Broadband backbones have the additional advantage of supporting many channels of communication. A single LAN can thus be used for computer to computer communications, terminal to computer communications, closed circuit television, and a variety of other uses. For this reasons, broadband backbones are sometimes installed instead of baseband backbones even for small networks.

HP supplies baseband networks. HP also supports broadband backbones through the use of recommended products from Ungermann Bass, such as the Buffered Repeater, which connects ThinLAN clusters to broadband backbones.

COMPANY-WIDE ACCESS MODULE



Although many engineering networks involve only a single site, there is often a requirement to connect engineering communities at several locations into a common network.

If the traffic between sites is not extensive, the simplest alternative is to use dial-up telephone lines and asynchronous modems. HP 9000 computers support standard UNIX communications services such as uucp and mail which utilize

these asynchronous modems.

In addition, the UNIX communications services can utilize X.25 networks through the use of an HP supplied X.25 multiplexer. The X.25 networks have the additional benefit of more reliable data transmission. In many cases they are also more cost effective than dial-up communications lines.

X.25 communications can be provided by public X.25 networks such and Telenet in the United States or Transpac in France. HP also provides switches which can be used to create a private X.25 network which would carry traffic for only a single company. Such a network may be of interest, for example, if there are special security or performance requirements. A private X.25 network also allows very tight control of network operations.

SUMMARY

HP AdvanceNet for Engineering provides comprehensive, standards based networking which meets the diverse needs of today's engineering community. Since HP AdvanceNet is based on standards, it provides a network which will evolve and endure for many years. Since it supports a multivendor computing environment, it offers flexibility in the selection of engineering computers.

HP is an active participant in the organizations defining future networking standards, such as the IEEE 802 committee, X/OPEN, the MAP Users Group, and the Corporation for Open Systems (COS). HP chairs several key working groups in these organizations.

As the requirements for engineering networks grow, HP AdvanceNet for Engineering will grow with them.

PRESENTING OUR IDEAS

DEREK NAGELS

Jack Austin Drugs Limited
Box 5013, Station 'A'
Downsview, ON, Canada
M3M 3E2

INTRODUCTION

"Listen to me!"
"I didn't mean that!"
"Why don't you understand me?"

How many times have we felt like screaming these expressions and more when we fail to get our point across. We've all had experiences that seem totally unfair. How many times have we lost an issue because we failed to present our viewpoint precisely? How may times have we observed some eloquent blowhard win his debate while we lose ours? If we're not satisfied, this presentation will help.

What we would like to focus on here is the ability that we have to triumph when we present our issues - correctly. This does not mean that we will never lose. It does mean that our ideas, our expressions, our opinions are at least listened to and considered.

In this business we must constantly examine countless innovative products, fresh ideas, novel concepts that invade our profession. We make an diligent struggle to remain enlightened, to keep abreast, otherwise we would not be here. So we acknowledge that our chosen occupation has changed dramatically. All you have to do is to compare this conference with Detroit to appreciate just how much change effects us. Change and the ability to adapt to it is the nature if this industry. We welcome this challenge. We may even criticize some of our compatriots who do not keep up with the times. You now, people who have not seen the need to change and keep doing things in the same old way. Now we ask ourselves -

"How are our presentations?"
"Are they the same style that we used years ago?"
"Do they reflect the advances that have occurred?"
"Or are there the tried old presentations?".

This paper, hopefully, will help to bring life and enthusiasm back into our presentations. Unless we make a

conscious effort to incorporate new ideas into our presentations the results will suffer. Tired, worn-out and stale may describe our proposals.

Before going further let's explain, this technique will aid us present our ideas for consideration. This will not provide us with a winning result if our proposal is flawed. Having the discipline to follow the procedures outlined in this paper will improve our chance to succeed. The obviously question is - "How can we create a winning presentation?" A successful presentation means diligent preparation.

If we want our presentations to succeed, the key ingredient is work. Don't expect to stroll into the 'board-room' unprepared and have people give your proposal serious consideration. It doesn't work. If we appear unprepared, how can we expect others to take us seriously? We should not expect others to give more consideration than our preparation indicates. It doesn't matter how good the original idea may be if the presentation is poor, the entire proposal is jeopardized. The greatest idea poorly presented will lose.

Don't just look at these ingredients as 'motherhood' and 'apple pie' issues. It is the application of these basic points that will make the difference. The difference between a winning proposal and one that is merely tolerated. This paper discusses, in depth, the following components of the formula:

Clear Vision
Gather Information
Re-evaluate Proposal
Proposal Concept
Proposal Layout
Re-evaluate Draft Proposal
Create Finished Proposal
Proposal Delivery

CLEAR_VISION

This clear vision suggests that we have a precise, clear image of what we intend to submit. Don't start the presentation until you have an 'over-all' plan in mind. I don't mean a detailed picture of all of the idiosyncracies of the concept. However, we must have a unobstructed view of the purpose of this idea.

We may discover that it's helpful to record the **objectives** of the idea. Write the **benefits** down, review them, change, add and delete. Keep this list update it until it satisfies

the objectives that we aspire to accomplish. Don't be unsettled if these goals change. We should expect them to change to include other advantages that are now very authentic but were not readily seen before. These additional benefits were not the motivation that started the idea. We may have a inclination to minimize and not give adequate consideration to these increased advantages. In fact it may be one of these expanded benefits that sells the concept and not the one that we started.

One of the problems is that to almost any predicament there many ways of achieving the desired result. There more than one way to skin a cat. We, obviously, must consider the resources at our disposal. We must justify any increase resource requirements resulting from carrying out this suggestion. We must be realistic in our selection of solutions - giving vigorous attention to the depletion of resources that we are proposing. Do we require more hardware, software, people, or any other resource. This ingredient is a essential component of the ultimate formula. Be realistic in these estimates. Remember upon acceptance of the proposal we must meet our commitments. You know what they say about most "IS" projects - they are late and over-budget. This pitfall is by using practical estimates.

We can not make a presentation that incorporates all of the conceivably solutions for the problem. Let's face facts we can't be all things to all people. Some may like our proposal. Some will wish we had chosen the other solution. We must make a choice and run with it. We have the responsibility, the integrity, the wisdom and the conviction to present the best technique to resolve the concern.

GATHER INFORMATION

At this period we should identify the direction of the overall concept. This gathering of information is the realm that is too frequently neglected. This demands persistance and perseverance to accomplish all of the tedious manipulates. The collecting of this data means work.

It usually appears that as soon as you have completed one viewpoint of this formula. Something else turns up making additional research a necessity. We will be tempted, at times, not to include some of these late comers into the formula. Nevertheless, if we aspire to champion the proper solution then we will explore and evaluate all conceivable methods. There is no 'quick and dirty' strategy that will generate a result of the high degree that we demand. Short-circuiting this fact-finding process we may invalidate the entire proposal. Make such to weigh all of the obtainable

possibilities. There will be times when our **gut-feel** doesn't work. Times when we examine the evidence accumulated and alter our concept. Be ready to change from some initial preconceived opinion.

The objective is to relate all expenses into tangible dollars. If dollars are difficult to determine, create a formula to show how the dollars were created. Some that require estimates must be conservative to have this document viewed a the work of a professional without any personal emotion. The gathering of information comes from the following areas:

Additional Resources
Maintenance Charges
Communication Charges
People Costs
Paper and Supplies
Training
Travel
Space Requirements
Saving

This is not an all encompassing list for rather a guideline that can we used.

Use some 'Spread Sheet' to record the information, updating and changing as required. This will form part of the support data in the final proposal. Create all costs on the same basis. I usually use monthly charges. Given a small degree of latitude we can create monthly charges for all costs. This includes all costs including 'one-time' charges.

RE-EVALUATE PROPOSAL

Thoroughly investigate the accumulated material. You will probably discover some surprising developments. The evaluation operation demands that we analyze all alternative solutions. We can now acknowledge the urgency for establishing all of the findings in dollars.

Establish a **Cost / Benefit** ratio for all the reasonable alternatives along with the cash requirements. Run the **cash requirements** though an amortization schedule based on the anticipated life of the product. The accountants are not exactly enthusiastic with this technique but it achieves results. Employing the **expected interest rate** we can establish a monthly cost for the purchase of these **cash requirements**.

The purpose is generate a document showing the various

solutions and exposing the cost and saving. The idea is to start a routine that will permit the decision making process with as little emotion as feasible. In other words we desire to deal with the facts. You will discover that we must desire to deal with facts and have as little emotion as possible. We must continually adjust the critical figures that will result in the **Cost / Benefit** ratio being modified. This is far from an exact science as the ingredients to the formula are in a rapid state of alteration.

Select the best possibility, perfect the components of the formula and we are ready for the skeleton presentation.

PROPOSAL CONCEPT

The objective is to develop a draft presentation in a manner that is appropriate for the final output without significant alterations. The more time we invest in the preparation of this blueprint, the less time spent during the construct of the final product. We incorporate minor adjustments in this document. This is not to infer that critical revisions will not occur, but rather, this should be well conceived.

This description should clearly demonstrate **the way to go**. Showing the best selection. With documented facts depicting why we have adopted that selection. There may be a temptation to list all options with no endorsement. This is an abdication of our responsibility, true, our mandate may have been to investigate the feasible answers to a specific condition. Obviously, we were appointed to head up this function, because people value our ideas. If we neglect to make recommendations, the presentation becomes merely a account of facts. Thus leaving the decision up to people who have less skill, knowledge and experience in that area.

The outline should display 2 or 3 of the choices. Typical the next best **Cost/Benefit** ratio along with any of the favorites that are open for consideration. These favorites should rate extraordinary attention explaining carefully and completely why they were not recommended. Complete this after a reasonable comprehensive investigation into the option. We can select the best solution this may not necessarily be the least expensive. There may be less costly alternatives than the one we selected. Offer explanations.

To aid the understanding of this analysis, record costing details to reflect the purchase for the next 5 years. This is helpful as we know factors such as hardware will decline while human resources will continue to escalate. Work these into your presentation. As we are not crystal ball readers, record all assumptions based on some other authority. "We

use a 5% wage increase according to our V/P Finance, Mr. Smith." Not only does this help to substantiate your figures it also eliminates some opposition.

The display of the material included in the **Draft Presentation** adheres very vigorously to the KISS principle- **Keep It Simple Stupid.** This is one of the key characteristics of a successful proposal. It's easy to comprehend. There is no room for the old **Baffle_them_with Bulk** routine. The objective is to get to the bottom line implications before someone pleads "How much?"

PROPOSAL_LAYOUT

There are many methods to layout the proposal. I will clarify the mode that I have determined is successful, for me. The first page summarizes the **Objectives,_Benefits, Cost,_Net__and_Conclusion**_and shows the recommendation. This document is easy to read and understand without compromising the details. The first page contains all the decision making points. With all the supporting information in the following pages. This method allows us focus the presentation on the benefits rather than the details. Document all substantiating details on subsequent pages of the proposal. Appreciate that the goal is to develop a document that is unclouded, easy to comprehend and dispenses complete facts.

Objectives			
	Option 1	Option 2	Option 3
Benefits			
Costs			
Net			
Conclusion			

The Net shows in dollars the benefits less the costs. Here also show the 'Cash Requirements' for the option.

Show the benefits and cost analysis for the various options. However, the conclusion must clearly illustrate the desired choice. Leave no doubt as to which is the best alternative. The conclusion is direct and to the point.

RE-EVALUATE DRAFT PROPOSAL

Attempt to read and criticize that Draft Proposal. Image someone that you are not enamoured with giving you the presentation. Find fault. Make changes until you are completely content with the result. Read the report as you were the decision maker. Is it clear? Can it understandable? Does the recommendation sound reasonable? Does the report drag? Be very critical of your work.

Try it out on your wife or husband, someone who has no little or no in depth knowledge. Discover whether they can grasp the substance that we are trying to communicate. We don't want to be untangling details that are unclear when we deliver the proposal. Complete all of the preparation. Try reading it through the eyes of the people who will be receiving the final presentation.

CREATE FINISHED PRESENTATION

Following the development and enhancement of the **Draft Presentation** to incorporate any new thoughts or clarifying the information. The final document should at a finished condition with only minors adjustments required. We need to acquire any covers, graphs or other presentation aids.

PROPOSAL DELIVERY

"Practice, practice, practice." You only get one shot so make it count. Polish the delivery in the sense that we have **in depth** perception, having appreciation all of the idiosyncracies of the proposal. However, Take care not to appear as a 'pitch' man so be cautious not to shine the material too much. We must retain ability to capture our and others enthusiasm.

CONCLUSION

The success or failure many times does not depend on **WHAT** we do as much as **HOW** we do. This framework can assist us to present our ideas. Showing techniques that will offer a challenge **Now and in the Future.**

Performance Problem Solving

Teresa Norman
Tymlabs Corporation
211 East 7th Street
Austin, Texas 78701

WHY SOLVE PROBLEMS?

Your ability to solve problems determines just about everything in your life: where you work, where you live, the car you drive, your hobbies, and, most importantly, how satisfied you are with your life.

Each day each one of us is called upon to fix something that's gone wrong, solve a puzzle, or invent something that does not exist. The dictionary defines a problem as "that which causes annoyance or difficulty". But let's examine a broader interpretation. For most situations in life, if the place you are right now is not the place you want to be, then you have a problem. Today there are many approaches to problem-solving which not only "fix what's wrong", but help you get where you want to be. They systematize the problem solving process.

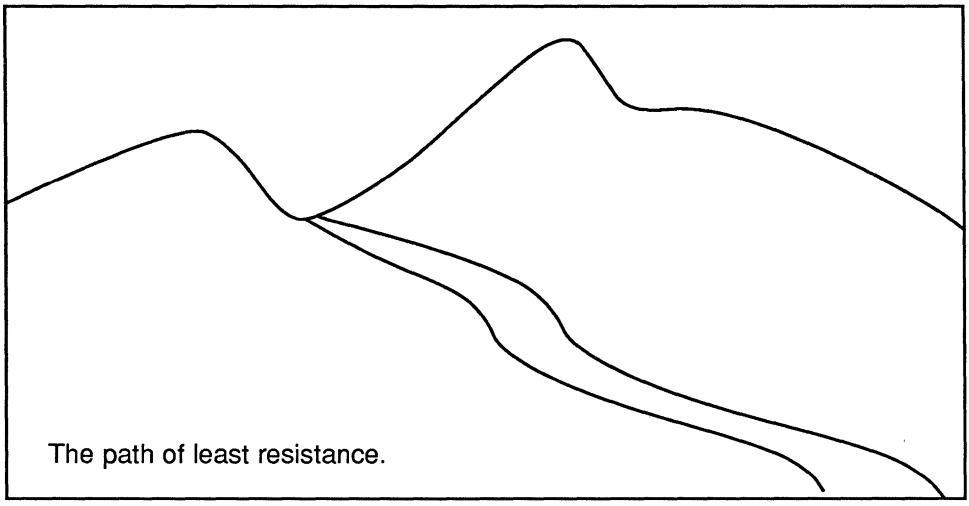
This process is known by many different names: decision analysis, framing (as in "framing" the problem and "framing" the outcome), lateral thinking, and goal setting. Each of these problem solving and decision-making methods follows the same path. And that path is to define precisely where you are, to define precisely where you want to be, and to define precisely the process for getting there.

I started out by saying that your skill at defining where you are, where you want to be, and how you will get there determines just about everything in your life. In a Harvard Business Review survey, a correlation was found between annual income and goal-setting practices. The respondents with the highest incomes had specific, written, and measurable goals. The respondents that needed financial assistance had no goals at all.

Edward de Bono, father of "Lateral Thinking" (the technique used to make the Los Angeles Olympics the first profitable Olympics), goes one step futher when talking about how your thinking ability affects your life. In his book, Six Thinking Hats, he states, "I once asked a group of very well educated Americans (graduate school) to give themselves a mark, out of ten, for their thinking ability. To my astonishment the average mark was eight out of ten. In other words, their horizons of what thinking could do were so limited that each person reckoned his or her thinking was almost as good as it could possibly be . . . people are remarkably complacent about their thinking - because they cannot conceive how it might be improved." In Venezuela, school children spend two hours each week developing their thinking skills in a program designed by de Bono and implemented under the Venezuelan Minister for the Development of Intelligence. As de Bono further asserts, "Being a thinker is a totally different self-image. It is an operating skill. You can do something about it. You can get

better at thinking just as you can get better at playing football or cooking."

This paper will explore several different problem solving methods and how they can be applied to what we all do: working with the HP 3000.



People, like rivers, take the path of least resistance. Fortunately, problem-solving methods function like our own personal Corps of Army Engineers, helping us change the underlying structures so the new path becomes the path of least resistance.

THE PATH OF LEAST RESISTANCE

People, like rivers, take the path of least resistance. We tend to follow the underlying structure in our lives the way a river follows its riverbed. To go against this requires tremendous will power or strain. And most often doesn't work. Fortunately, problem solving methods act like our own personal Corps of Army Engineers -- they help us change the underlying structures so that moving in a new direction becomes the *new* path of least resistance. In this we move from willpower to creative power to answer the question "What results do I want to achieve?"

Robert Fritz, founder of DMA and the Institute for Human Evolution, states in his book, The Path of Least Resistance, "As you develop any ability you need to expend less and less energy, and as you increase your facility in using any ability, you begin to master the use of your own energy in creating what you want. As you develop willpower, however, you need to expend more and more energy when you use it." This of course applies to any ability - skiing, cooking, building a computer, managing a company, or solving problems. As we mentioned, our goal is to ease the process and cause the river to flow in the direction of solving our problems.

GOOD AS GOLD: ATTITUDE

It is true that we are either the victims or the beneficiaries of our attitudes. But, why would problem solving start with this particular topic? It has been said that the difference between a professional musician and an amatuer is that the amateur plays well when he or she feels well and the professional musician plays well irregardless of how he or she feels. The same can be said of the problem solver. And I think each of our experiences bears witness to this. The truly superb problem solvers I know in data processing create their own reality each day. They are the outstanding programmers and managers who consistently turn out predictable results.

Robert Fritz, as cited above, states, "If your emotions become the dominant factor in your life, the power in your life becomes "how I happen to feel," not "what I truly want." In contrast, if the power in your life lies in what you choose, you are reunited with your real human power, and the way you happen to feel becomes subordinate to what is actually more important to you."

Another aspect of this shift in thinking is time consciousness. If you don't have to wait to feel good, you are free to solve your problem now, not at some undefined point in future time. Time consciousness is also deciding in advance how much time you will allocate to solving a problem. Not setting a deadline is like handing out blank checks. It allows a problem to determine how much of your time it will take. If you have erred in your initial calculation, the time frame can be adjusted, but you cannot retrieve lost days, weeks, months, or years.

Tomorrow and tomorrow, and tomorrow ...

When will you solve your problem?

Not setting a deadline is like handing out blank checks. It allows the problem to determine how much of your time it will take.

GETTING STARTED: WHERE AM I?

When I was a small child my father used to repeat to me the title of his favorite sermon, "The right way to begin is to begin the right way, right away." In all the problem solving methods available each one has techniques for defining your starting point: getting started in the right way. The framing approach, as described by McMaster and Grinder in their book, Precision, uses a series of questions to pinpoint the starting point with more accuracy than we usually receive in business communications. For example, let's say you are faced with

the following directive: The computer is slow - do something about it. The framing technique would not recognize such a vague and messy communication as an acceptable starting point to begin a solution frame. The framing technique suggests a series a "blockbusting" questions to go from low-quality information to precise, high-quality information. Using the McMaster and Grinder Precision Model we might challenge this with the following questions:

The computer is slow.

What is slow?

The on-line programs are slow.

How slow are the on-line programs?

The users sometimes wait a long time between transactions.

What is the computer doing while the users are waiting?

Updating the data base.

Updating only?

Wells, no, actually, any kind of file accesses: reading, updating, etc.

Let me get this straight, any time an on-line user process is accessing a file of any type, they wait a long time?

Right. Any file. Wait a long time.

Would you agree that we must begin looking at the disc activity on our systems?

At this point you might pause to do some investigation. Using any tools you have available you could research what is happening on the system to determine if you had a data base structure problem, or a problem within a particular application, or were just disc I/O bound. (An excellent example using just this same starting point, *the computer is slow*, is "Ten Questions To Ask A Slow CPU" a paper by Bob Green of Robelle.)

When you have achieved a precise starting frame, the next step is to develop just as precise and verifable a desired state or outcome frame. According to McMaster and Grinder, "One of the most frequent uses of the Outcome Frame is to remind participants of a group to orient specifically to the Desired State they are attempting to achieve." In other words, to put yourself into the solution, to establish some momentum towards solving your now carefully-defined problem.

> **Most folks aim for nothing in particular, and seem to hit it with amazing accuracy.**
>
> **Henry Ford**

The same blockbuster questions can be applied to the outcome as were applied to defining the present state. In other words, quantifying and qualifying where we want to be.

Using the above example, let's assume that our investigation of the slow computer revealed that it was disc I/O bound, in other words, there were so many pending requests for disc activity that each on-line process disc request was lining up to wait. Further, we had determined that neither the application program itself nor the data base structure was at fault. With the two obvious causes eliminated, we are challenged to look in other directions.

At this point, most problem solving methods advocate generating as many solutions as possible. This creative process frequently goes by the name of brainstorming. Brainstorming is the generation of new ideas and new approaches regardless of any practicality, merit, or feasibility. The winnowing out of the "good" ideas from the "bad" comes later in the process when the actual path to the outcome state is being constructed.

Edward de Bono, who has written many books on the generation of new ideas, states that both creative thinking (which he calls *lateral thinking*) and our traditional thinking (which he calls *vertical thinking*) are necessary and have their place. In his book, Lateral Thinking, he states, "The deliberate generation of new ideas is always difficult. Vertical thinking is not much help, otherwise new ideas would be far easier to come by. Indeed, one would be able to programme a computer to churn them out. One can wait for chance or inspiration or one can pray for creativity. Lateral thnking is a rather more deliberate way of setting about it."

For our slow computer, we might employ some of these techniques to focus on non-program areas for disc I/O improvement. Our brainstorming session might produce ideas like routing all disc requests through one "I/O" program, eliminating all batch jobs that compete with the on-line programs, staggering work hours, reducing unnecessary log-ons (which require many disc I/Os to log to system tables), allocating commonly used programs to reduce the table logging that programs do when they are first run, properly segmenting programs to reduce memory manager disc activity, etc.

As we stated at the outset, the purpose of this step is to increase the pool of possible solutions. How workable any particular idea is can be determined at

the next step when we decide where we are going. Will we opt for speeding up the slow computer to the point that the on-line users no longer complain, will we try for a certain percentage improvement, or do we need more information? The blockbuster questions - how much improvement, where to go for improvement, when will we know we've sufficiently improved,etc. - all need to be considered.

WHERE AM I GOING: THE PATH AND THE MAP

Many carefully defined solutions with clear goals and stated outcomes never see the light of day because the map to get from present state to desired state is never constructed or is never implemented, or is never monitored.

STATE 1: The problem is carefully defined, probable causes and solutions are generated, but nothing is ever decided. This is analogous to sending away for the travel brochures and checking the flight schedule, but not making any reservations. Many problems just get a lot of air time and then fall into oblivion for many "lack" reasons: lack of enthusiasm, lack of money, lack of time, lack of resource. One possible solution is to see the "lack" itself as a problem in and of itself to be solved prior to continuing.

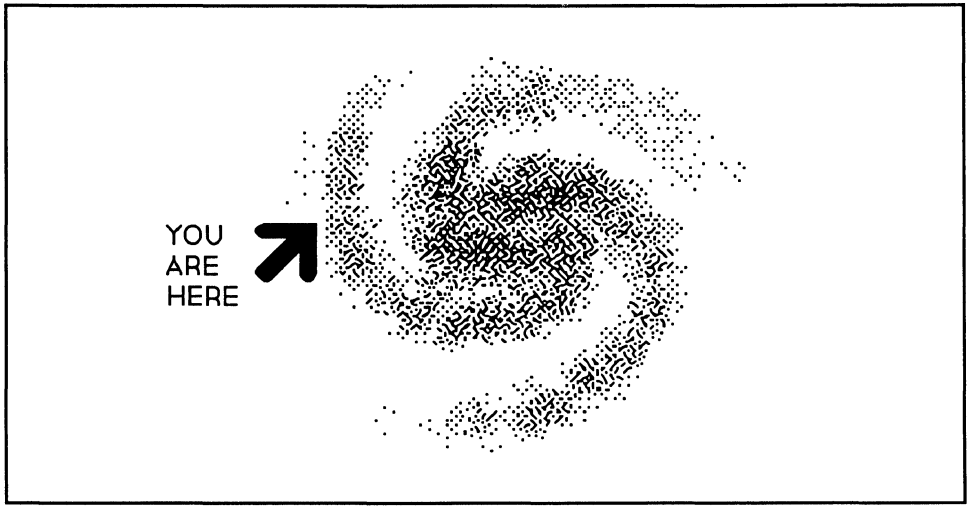
STATE 2: What if you made the travel reservations but kept deviating from your itinerary? I know many people who vacation quite nicely this way, but deviating from the established problem-solving path is a trip to oblivion for your problem's solution. Monitoring is necessary to permit deviations where they are necessary to keep on track, but to prevent deviations which occur for reasons not at all related to the solution of the problem.

In our slow computer example, the monitoring step might involve establishing test criteria and anticipated performance gains, then performing the tests and comparing the results to what you expected. Remember, this is your problem and the responsibility rests with you.

IN CONCLUSION

When you solve a problem, you are participating in a process as old as our species: using your mind to affect your environment. Albert Einstein said, "Each person has at least one idea in his or her life that could change the world." And the difference between genius and non-genius has to do with whether those ideas are ever implemented.

Our daily work and personal life are the source from which these inspirations come. And that makes the problem solving process a quest. Edward de Bono states, "Thinking is the ultimate human resource. Yet we can never be satisfied with our most important skill. No matter how good we become, we should always want to be better."



Perspective is knowing how your problem fits into the grand scheme of things.

BIBLIOGRAPHY

Conceptual Blockbusting, James L. Adams (Addison-Wesley, 1974)

Six Thinking Hats, Edward de Bono (Little, Brown, and Company, 1985)

Lateral Thinking, Edward de Bono (Harper & Row, 1970)

The Path of Least Resistance, Robert Fritz (Stillpoint Publishing, 1984)

Precision: A New Approach To Communication, Michael McMaster and John Grinder (Precision Model, 1980)

IMAGE

THE FUTURE AS SEEN THROUGH GROUCHO'S GLASSES

By Terrence D. O'Brien
Dynamic Information Systems Corporation

January 1st, 1992. Think about this date for just one moment and try to visualize what your computer system and the software running on it will look like five years from now. Will accounting, manufacturing, distribution, and other applications look and work the same or will new technology in equipment and software change the way we store, process and retrieve information?

Future planning for the Information Services Center (a preferred name for the Data Processing Department) is an absolute requirement of the data processing professional who must keep abreast of both changing technology and evolving information requirements so that future applications will be a harmonized blend of cost effective solutions. Five years is a reasonable period of time for planning because systems and applications designed today and implemented over the next few months to two years will still be in use in 1992. On the other hand, planning beyond five years is probably futile since new advances in technology will certainly have a major impact on any systems that are developed after 1992.

Technology that will be part of any future application needs to exist today or at least be proven before being seriously considered. This is the only viable choice since so many new ideas in hardware and software never make it into commercial use, and until new technology becomes tested and used, the true performance and functionality is unknown. Therefore, this five year projection will be based solely on existing or proven equipment and application tools.

THE NEED

The applications being planned or built today will require a new level of sophistication to meet an expanding and better educated user base. The personal computer (PC) has had tremendous impact on the Information Center, less so for the technical ramifications, but more so because it has opened up computer processing power to more users and allowed those same users to become computer literate. In five years, a new breed of middle executive will use the computer training received in school as a platform for demanding more from data base systems. He will have been spoiled with fast access on small data bases, on-line context sensitive help, concurrent processing of multiple tasks and standardized interfaces. He will be unhappy with software that costs thousands of dollars to develop and with running on a half-million dollar "mainframe" that does not meet some of the emerging standards developing in the PC marketplace.

Not only will there be a need for better existing systems, but more information retrieval will be expected from them. The old promises of the "Management Information System" will be due and payable in five years and the Information Center will need to provide instant on-line access to large volumes of information. This demand must be met with today's environment, tools and techniques since waiting for HP or any other vendor to provide a future solution is far too risky.

THE OVERALL ENVIRONMENT

In five years, the environment for data base applications will be slightly different from today. Few HP3000s will be running general applications such as word processing and spreadsheets. The HP3000 is one of the best (if not the best) transaction oriented data base systems available, but it is not well suited for computation intense applications such as spreadsheets. Hewlett Packard has recognized this and made a decision not to provide spreadsheets, graphics, and word processing packages for the new Spectrum line of computers. These applications are much better suited for a micro or personal computer system where better software and faster processing is available. The HP3000 will become dedicated to data base applications, communications processing (electronic mail and remote service bureaus), and PC file sharing. Note that HP did not make an initial error in originally providing spreadsheets, graphics and word processing on the HP3000. At the time HP provided these tools, they were cost effective to process on the 3000, but with the continued slide in PC prices coupled with a continued increase in horsepower, the PC has emerged as the preferred and cost effective choice.

Although HP's new Spectrum line promises to provide faster processing speed, it will have little other impact on how the data base applications look and feel to the end user. A faster CPU can make certain operations quicker, such as printing a general ledger in 30 minutes instead of one hour. Although more platable, it will not change the overall functionality of the application. Additionally, Spectrum must fall into the class of unproven technology even though HP points to the success of the Unix based 840 as an endorsement of HP's Precision Architecture. HP has only currently proven that scientific based systems with a simple operating system (Unix) will work very well with the new architecture. However, this does not necessarily mean that it will work as well with commercial applications using a complex operating system such as MPE/XL.

The announcement of the new, powerful and inexpensive Micro/3000 will also have an impact on our data base environment because data base systems will not be competing against CPU and disc I/O intensive application development. It is now more cost effective to purchase a separate development system for the Information Services Center.

HP's new inexpensive and large capacity Eagle drives will have a more profound impact on the data base environment. Every time the cost per byte for disc storage drops, it becomes more cost effective to store additional information or more historical data. What was cost prohibitive five years ago, becomes possible today. Data bases in five years will be bigger in both the type of information kept and the volume of historical information maintained.

The terminals in use today will essentially be the same as those in use in five years. They will likely be cheaper and thus allow more casual users to have access to the corporate data bases. However, most information users (as opposed to information providers, e. g. data entry operators) will tend to use PCs as the primary work station and communicate to the HP via terminal emmulation software. There will be more of a demand to access the expanding information base with direct access from the PC's spreadsheet rather than the tedious batch processing indicative of systems such as HPACCESS. Oracle, a supplier of data base and fourth generation tools on other hardware, has already developed and proven that a direct spreadsheet to data base interface is possible and provides an easy-to-use data base interface for the end user.

A tool to allow Lotus 1-2-3 direct access to IMAGE data bases will certainly become available within the next five years. This will have a dramatic effect on future applications because it will easily become the preferred interface for information retrieval systems.

HOW APPLICATIONS WILL LOOK

Traditional applications such as accounts payable and order entry will look very much the same in five years with one notable exception. Additional on-line and flexible retrieval of data will have to be added to allow a new level of user friendliness. The retrieval of customer records via name and descriptive information such as address or contact person will be required. Retrieval of a part record or stock records via a portion of a description will continually be requested by users. These types of retrieval options will be commonplace in five years because the technology to perform such tasks is available today and being implemented by several companies including software development houses to add increased user friendliness and functionality to their applications.

These same applications will also be improved by the availability of windowing software, should HP or a third party be able to deliver the capability to remain in one application while running any number of other programs. Although this is currently unproven in the HP world, windowing software is having a tremendous impact on the PC environment and has been shown to provide increased user satisfaction. A windowing environment allows the user to work in one application such as a data base inquiry and immediately jump into the electronic mail system to send a quick message and then instantly return to the original application. Windowing's greatest impact however will be the ability to allow more flexible and faster retrieval on existing applications. Through windows, applications written today can be updated to handle future retrieval requirements without rewriting the original software.

Unfortunately the functionality of windowing software will be impacted by the display rate of existing terminals. The current top speed of 19,200 baud will need to be increased to at least 56 kilobaud before windowing software on the HP approaches the same functionality as the personal computer.

Applications oriented primarily to information retrieval such as sales history analysis and accounting systems will see the most dramatic changes over the next five years.

There will be an increased need for better and faster retrieval on the growing volume of information maintained. Users will require access to their data quickly using multiple selection criteria which may span several fields or even several data sets. These information based systems will be the primary focus for new system development since traditional operational systems will be in place.

THE DATA BASE MANAGEMENT SYSTEM

In five years the primary data base management system (DBMS) in use on the HP3000 will still be IMAGE. And applications designed today and still operating in 1992 will be using a 20 year old data base. This is because there is a wealth of software tools, applications, and knowledge that has been developed around IMAGE which far exceed any other DBMS offering. This is also because HP's future ALLBASE software must still be classified as unproven new technology which could be delayed

and suffer unforeseen performance and integrity problems. No one will know for sure until the product is released and used in several high volume and multi-user applications. Until that time, IMAGE is the only viable DBMS choice today for future applications on the 3000.

IMAGE and its add-ons (restructuring tools, high speed serial reads, report writers, high level languages, and additional index structures) are also far beyond HP's current relational data base offering - HPSQL. SQL simply lacks both the high level language and more importantly the indexing and retrieval options to meet current and future application needs. SQL (and HP's future ALLBASE DBMS) and most other relational data bases rely on a KSAM like B-tree structure to handle indexing. This structure alone is not powerful enough to handle fast retrievals across multiple fields or retrieval by individual words or combinations of words within a record (e. g. selecting parts based on portions of the description field). IMAGE, with its fourth generation languages and retrieval enhancement tools, will be the choice for most future applications, not because of SQL's reported performance problems, but more so because SQL lacks the functionality required. Without multiple field retrieval, SQL is poorly suited for operational systems such as accounts payable and even worse for informational systems such as sales and prospecting or accounting.

WHAT THE FUTURE WILL BRING

The future will bring friendlier software with more accessible information and better retrieval options. Not because of any revolutionary change in the hardware environment or the data base system, but because of a gradual evolutionary change to the existing applications. January 1, 1992 is just around the corner and IMAGE, at age 20, will be grown up and ready to handle the future.

EXCERPTS FROM

A PRACTICAL GUIDE TO DISASTER RECOVERY PLANNING

Copyright 1986

Business Recovery Systems, Inc.

All Rights Reserved

EXCERPTS FROM
A PRACTICAL GUIDE TO DISASTER RECOVERY PLANNING

by

Michael J. O'Malley, CPA and Raymond J. Posch, CDP
Business Recovery Systems, Inc.

According to industry sources, a business burns ONCE EVERY FIVE MINUTES in the United States. Ninety percent of these fires result in the irretrievable loss of critical business records. OVER FORTY PERCENT NEVER REOPEN their doors as a result. Fourteen percent suffer a reduction in credit rating. These are frightening statistics.

COULD YOUR ORGANIZATION RECOVER FROM A DISASTER?

What would your company do if its computers were destroyed by fire? Would you still be in business for very long? How much business, customer goodwill, or CASH FLOW would be lost...assuming that you could recover?

How would a severe earthquake, a bombing, or the act of some disgruntled programmer affect your computer service? How long would it take to replace the computer room, air conditioning, power equipment, cabling, communications equipment and computers? Where would you locate key personnel? What would the delay cost in lost production, regulatory penalties and the inability to bill for work previously accomplished? When would your operation be normal again?

ONLY THE PREPARED SURVIVE MAJOR BUSINESS DISASTERS

Recent studies have shown that companies lose most essential business functions within 3 days of a computer disaster. Within 10 days they would be technically dead without backup. Statistically speaking, you will not recover as a business from a major disaster unless you have a realistic, tested, disaster recovery plan. The odds against the unprepared are overwhelming. Unfortunately, less than 20% of the businesses which depend on computers have a disaster plan. Can your business afford the gamble?

DISASTERS HAPPEN TO ALL OF US

The easiest thing to do is nothing. Play the "odds". Gamble that a disaster could never happen to "us". In reality it could happen to any of us tomorrow. We have compiled from several trade publications a short list of organizations which have experienced major disasters:

SANTA BARBARA EARTHQUAKE - 12 companies experienced major computer outages.

TERRORIST ATTACK ON ITALIAN MOTOR VEHICLE MINISTRY - The entire vehicle registration data base was destroyed. It took 18 months to restore.

DISMISSED EMPLOYEE AT PAYCHECK, INC. - An employee broke into the computer center over a holiday weekend and destroyed all the company's disk packs including backup copies. It took two weeks and countless man hours to manually reenter 95,000 employee payroll records.

BURST WATER VALVE AT TEMPLE UNIVERSITY - Water flooded the computer room at Temple University. It took days to restore the center back to operation.

FIRE AT GENERAL COMPUTER SERVICES INC.- Fire gutted the data processing center, completely destroying the company's computers. GCS's Disaster Recovery Plan was initiated and the company was able to restore its operations in an orderly manner.

FIRE AT NORTHWESTERN NATIONAL BANK - Fire destroyed the 16 floor building which housed central operations for its 86 affiliates and 500 correspondent banks. Northwestern became operational agai chiefly because of a well-prepared Disaster Recovery Plan.

10,000 GALLONS OF WATER AT MAZDA - The roof of the building housing MAZDA's data processing center collapsed during a heavy Pacific rain storm and demolished the company's computers. The disaster brought to an abrupt standstill data processing for MAZDA's 31-state business.

FIRE AT A MAJOR INSURANCE COMPANY - As the Company entered the final stages of its conversion to a new computer, a weekend fire destroyed the firm's headquarters building which housed its computer facility. The firm's computers were completely unusable.

EXPLOSION AT WELLS FARGO - A propane tank exploded in the basement of Wells Fargo offices in Denver on New Years Day 1984. The fire quickly spread through the office building destroying everything in its path.

A DISASTER RECOVERY PLAN IS INSURANCE!

The reason we buy insurance is so we will be covered in the relatively unlikely event of a major loss. Chances are, our homes will not burn down tonight. But the consequences can be so significant that we must carry insurance on our homes. Likewise, we cannot afford to ignore the consequences of a major disaster in our business. Disaster Recovery Planning is the principal component of BUSINESS DISASTER INSURANCE!

YOUR DISASTER RECOVERY PLAN

Most DP Managers, Internal Auditors and users agree that disaster recovery planning is something that should be done. However, few organizations actually have any kind of formal plan. Why is this?

The reason is really quite simple -- a Disaster Recovery Plan takes a considerable amount of time and effort to develop. It is a difficult process. It is hard to know just where to start and our day-to-day responsibilities often leave us with little time to devote to such a major undertaking.

What is Disaster Recovery Planning? It is planning for potential fires, floods, accidents or other calamities that may befall your organization. The objective of the planning project is to prepare a plan which provides for the continued operation of your data processing facility or other functional areas of your business in the event of an emergency. At minimum, the plan should address central computer operations. Optimally, it should address all aspects of the business including manual and automated functions.

The purpose of a Disaster Recovery Plan is to increase the chances of survival and to decrease the amount of loss. It identifies how the critical computer-dependent services of the business will be restored to operation with the least disruption. The best plan will be simple, specific and well thought through. Remember, the better the plan, the better your insurance.

START WITH COMPUTER CONTINGENCY

To do a full-fledged Disaster Recovery Plan which addresses all aspects of the business(see figure 1), from telephone service to paper flow within all departments, would be an extremely large undertaking. It would require the active involvement of your entire organization.

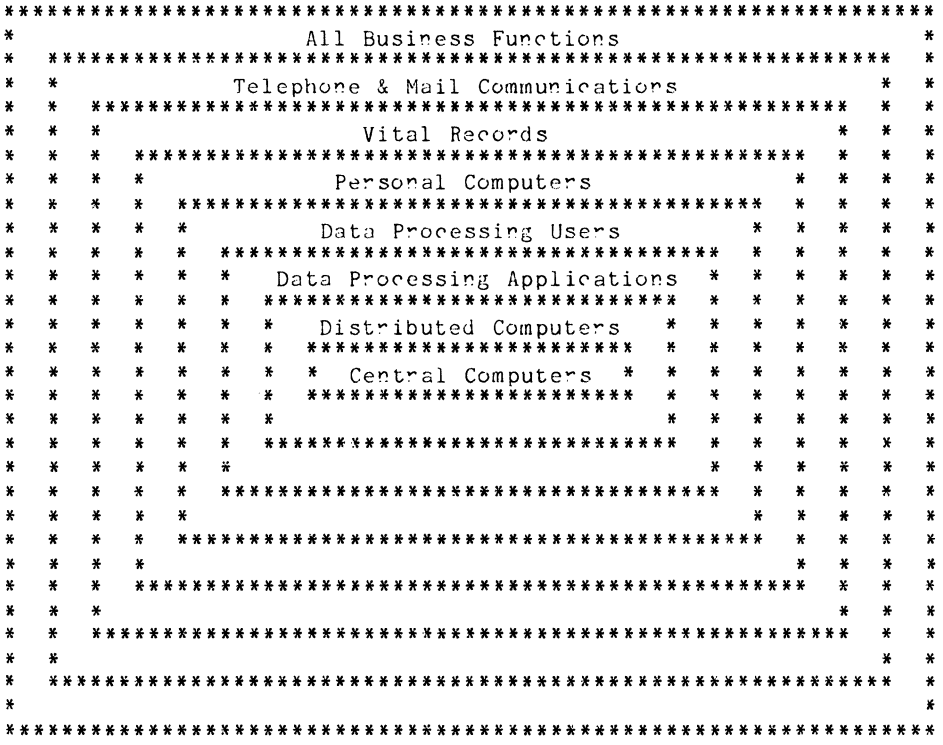
We advise starting with a more manageable piece... the critical areas of the central computer operation. Initially limit the Disaster Recovery Plan to COMPUTER CONTINGENCY. Develop the smaller plan first and if you later have the opportunity, expand it to address other business considerations.

To help managers get started, Business Recovery Systems has prepared "A PRACTICAL GUIDE TO DISASTER RECOVERY PLANNING". It includes a suggested outline for the Disaster Recovery Plan document, a workplan for developing the Plan and a discussion of the steps that must be accomplished. This excerpted version of the GUIDE does not include the project workplan. The workplan may be obtained from the authors upon request.

When you have finished with this Guide, proceed to organize your disaster recovery planning project. You will probably want to include many of the following steps.

FIGURE 1

LEVELS OF DISASTER RECOVERY PLANNING



STEP 1 - GET STARTED!

Because having a Disaster Recovery Plan is so important to your organization's ability to stay in operation following a disaster, put it on your agenda now and start work as soon as possible. The sooner you begin the better! Should a disaster strike next month, you might not be totally prepared, but you will be better off than if you do nothing today.

If the man-hours needed to do the plan are limited, then limit the scope of the Disaster Recovery Plan and simplify the project. The important thing is to have some plan rather than none at all. Of course, if you take this approach you will certainly want to beef-up the Plan in the future, as time allows.

Getting started means identifying the scope of the project you are willing to undertake and the resources you are willing to commit to the disaster recovery planning process.

STEP 2 - OUTLINE THE DISASTER RECOVERY PLAN

In order to begin developing your Disaster Recovery Plan, you must prepare an outline of the Plan. Your outline should be consistent with the scope of your project and available resources you identified in Step 1. Included with this Guide is a suggested outline (see table 1) that you should find helpful in this effort.

At this point , you should consider how the completed Plan will be printed and distributed. You should also note that disaster recovery plans, in order to remain viable, must evolve as the business itself changes over time. Most managers will want to develop and maintain the Plan on a computerized word-processing system. Since the outline is the beginning of this document preparation, this is a good place to resolve these issues.

DISCUSSION OF THE SUGGESTED OUTLINE

SECTION 1 - OVERVIEW

The first section should provide a general overview of the Disaster Recovery Plan. Be careful to communicate in clear precise language. This section should include statements of purpose and policy, a brief description of the contents of the Plan and a discussion of your Readiness Team.

The Readiness Team will consist of a group of people from within your organization. The Team will be responsible for "first response" actions in the earliest stages following a disaster. In outlining this group's makeup and responsibilities, you should include a statement of purpose, define the position of Emergency Coordinator, describe the general process of what should happen in an emergency and who the responsible players are.

SECTION 2 - MAJOR SERVICES, USERS AND KEY CONSIDERATIONS

The major services provided by your computer systems should be documented in this section of your plan. Summarize the nature of your critical applications. Document the users of these applications, critical work schedules and dependencies that may be inherent in processing these applications. Prioritize these major services. Not everything may be recovered immediately in an emergency if the company experiences a loss of processing capability. The information in this section will be used to help establish priorities and consider options for assigning resources.

This section should be organized by system. Each specific service or system should be available for quick and direct reference. Examples of major services are: Payroll, Billing, Production Scheduling and Order Entry.

SECTION 3 - POTENTIAL INTERRUPTIONS AND GENERAL PROCEDURES

There are two levels of recovery procedures which you probably want to address in your Disaster Recovery Plan: (1) a "major" disaster - with detailed procedures for switching operations to a contingency computer site and (2) lesser emergencies, or simply interruptions of service, that may or may not require switching to a contingency site. This latter set of procedures will be more generalized, since you cannot prepare for an unlimited number of possible interruptions of service which might occur.

This section of your plan should address lesser emergencies and interruptions. It should be organized by the most serious and/or probable emergencies which might occur, such as: minor fires, power outages, telecommunication failures and hardware failures. Included for each type of interruption should be the GENERAL procedures for handling the emergency, along with the detailed actions which vary from situation to situation. Your stated objective may be to operate in a less-automated fashion while restoring normal service as soon as possible.

SECTION 4 - POLICIES FOR REDUCING RISKS

When you have identified potential interruptions of service, as you did in Section 3, you are ready to define policies for reducing risks relative to those emergencies. You may have most of these policies already in place. The purpose of this section is to document those policies for easy reference.

This section should be organized as a series of policies, such as protection of data, protection of facilities, offsite forms storage, insurance, etc. Additionally, for each policy, two kinds of risk reduction should be addressed: (1) reducing the risk of the interruption occurring and (2) if it does occur, reducing the risk of an inadequate recovery process.

SECTION 5 - CONTINGENCY SITE DESCRIPTION

This section should include a description of your contingency computer site for major disasters affecting your computer operation. It should state the location, names of contacts and telephone numbers. It should describe the contingency hardware, software and facilities. It should describe the accommodations for your organization's staff, scheduling considerations, limitations and procedures.

Business Recovery Systems provides this information to its disaster recovery hot-site clients. You should be able to obtain the necessary information from any other contingency site that you select.

We should point out that selecting a contingency computer site is a VERY IMPORTANT part of the disaster recovery planning process. Some organizations go the route of developing a mutual backup arrangement with other organizations in the same city or general vicinity. We believe that very few if any of these reciprocal agreements are really workable, so be careful with such an approach. How many businesses would really be willing to severely limit their computer functions for another organization which has experienced a disaster? Alternating schedules are generally not feasible because of conflicting configuration requirements and the substantial amount of time required to reload and reconfigure. What site could possibly have so much excess disk space to run two systems side-by-side?

Two alternative approaches are available: hot-sites and cold-sites. A hot-site provides an unused computer which is ready and waiting in a working facility. A cold-site provides only the facility and not the computer, in which case you have to obtain the computer from the manufacturer or some other source. We recommend that you choose a hot-site if at all possible. It may take weeks to order and install the necessary hardware to recover your business using the cold-site approach.

SECTION 6 - RECOVERY PROCEDURES FOR A MAJOR DISASTER

Now comes the main part of the Disaster Recovery Plan. You should by now have defined a general Readiness Team, defined general procedures for specific kinds of service interruptions, and identified a contingency computer site. Knowing these things, you can develop specific procedures for a major disaster that requires switching computer operations to a contingency site due to destruction of the data center or other factors.

This section should include definitions of Emergency Action Teams and their responsibilities, procedures for activating contingency site operations, specific procedures for data processing operations and interface requirements for user departments. It should also include procedures for replacement of the data center and return to normal operations. This will be a lengthy portion of the Disaster Recovery Plan and should be organized as a series of procedures listed by name in the table of contents.

SECTION 7 - TESTING AND MAINTENANCE OF THE PLAN

This section should contain policies and procedures for testing and maintaining the Disaster Recovery Plan. There is no assurance that the Plan will work unless it has been tested. You must be confident that the computer contingency site is workable, that the procedures do not have technical or other unforeseen problems and that key people understand the Plan both in the general and specific cases.

Just as important, the Plan must be reviewed and updated on a regular basis. The Plan must evolve in parallel with your organization, since very little about a business remains static for any length of time. It must be reviewed and updated for organizational, technological and business environment changes that may have occurred.

APPENDICES

Finally, the Disaster Recovery Plan should have attached as appendices all additional sources of vital information which may be needed to carry out the recovery procedures. Examples are directories of employee names, addresses and phone numbers, hardware and data center configurations, and so on.

STEP 3 - PREPARE A WORKPLAN

Having a project plan is always important to a project's success. Spend the upfront time to prepare a project plan, including making estimates of the resources that the project will take. An example workplan is not included with this article, but may be obtained from the authors upon request. It will be a good starting place; but you must refine it for your organization.

A workplan breaks the project down into tasks and steps. For each task and step, levels of effort should be estimated in man-hours or man-days. Having determined your planning project scope, the probable project team and the outline of the plan to be developed, you should be able to estimate work effort and prepare a schedule to accomplish each task.

The workplan generally follows the outline of the Plan. The workplan defines actual steps which must be performed, while the outline lists specific services, interruptions, policies, procedures and appendices not necessarily itemized in your workplan. You may wish to expand your workplan to include more detailed sub-steps in order to accurately estimate the resource requirements. Note that a typical workplan consists of two parts: (1) a detailed workplan which shows the steps within each task and (2) a summary workplan which lists the tasks only (detailed steps are not shown). The detailed workplan is necessary to plan specific work which must be accomplished; the summary is useful for a high-level project view.

STEP 4 - CONDUCT THE PROJECT ACCORDING TO YOUR WORKPLAN

Now you are ready to begin the real work of the Disaster Recovery Plan. As you should now realize, the project consists of analysis, planning and documentation. Your project team needs to consist of good, general analysts. Your team may well all be managers.

As you no doubt know, project plans usually are not perfect. Things come up that change your expectations, the sequence of tasks and your time estimates. Proceed as any good project manager would by being adaptable and remembering the real goal -- to develop a Plan that will help your organization survive a computer disaster and keep its business successful.

Michael J. O'Malley is President of Business Recovery Systems, Inc. Mr. O'Malley has over 12 years of experience working with data processing management and users. Mike is an MBA and a Certified Public Accountant in the State of Colorado.

Raymond J. Posch is a Senior Manager for Business Recovery Systems, Inc. His responsibilities include Recovery Planning and Customer Support. Mr. Posch has over 17 years of data processing, operations and planning experience. Ray is a Certified Data Processor.

Business Recovery Systems, Inc. provides contingency Hot-Site and other Disaster Recovery services. A complimentary GUIDE may be requested by calling 1-(800)654-2493.

TABLE 1

SUGGESTED OUTLINE

DISASTER RECOVERY PLAN

1.0 OVERVIEW

1.1 Objectives
1.2 Overview of the Plan
1.2.1 Policy Statement
1.2.2 Contents of the Plan
1.3 The Readiness Team
1.3.1 Purpose
1.3.2 Organization and Planning
1.3.3 Emergency Coordinator
1.3.4 Alternate Emergency Coordinator
1.3.5 Offsite Emergency Coordinator & Alternate
1.3.6 The Use of Emergency Action Teams
1.3.7 Emergency Control Center
1.3.8 Management Succession

2.0 MAJOR SERVICES, USERS AND KEY CONSIDERATIONS

2.1 System #1
2.2 System #2
. . .
2.n System #n

3.0 POTENTIAL INTERRUPTIONS AND GENERAL PROCEDURES

3.1 Fires
3.2 Electrical Power Outages
3.3 Telecommunications Failures
3.4 Hardware Failures
3.5 Software Failures
3.6 Applications Failures
3.7 Major Disasters

4.0 POLICIES FOR REDUCING RISKS

4.1 Protection of Computer Data
4.2 Protection of Data Center Operation
4.3 Protection of Vital User Records
4.4 Backup of Hardware, Software, Supplies & Forms
4.5 Insurance

5.0 CONTINGENCY SITE DESCRIPTION

5.1 Contingency Site Location and Contacts
5.2 Machine Configuration and Facilities
5.3 Accommodations for Data Processing Staff
5.4 Accommodations for User Departments Staff
5.5 Scheduling Considerations

6.0 RECOVERY PROCEDURES FOR A MAJOR DISASTER

6.1 Emergency Action Teams and Responsibilities
6.2 Notification of Emergency Teams
6.3 Notification of Contingency Site Provider
6.4 Activation of Contingency Operations
6.5 Specific Procedures for Data Processing Operations
6.6 Specific Procedures for User Department Operations
6.7 Procedures for Replacement of Data Center
6.8 Procedures for Return to Normal Operations

7.0 TESTING AND MAINTENANCE OF THE PLAN

7.1 Policies and Procedures for Testing
7.2 Policies and Procedures for Review and Update

APPENDICES

A. Emergency Team Lists: Names, Phone Numbers & Addresses

B. Data Processing Organization Chart

C. Data Processing Directory: Names, Titles & Phone Numbers

D. User Departments Directory: Names, Titles & Phone Numbers

E. Vendor Lists: Names, Numbers, Addresses & Descriptions

F. Service Agreements

G. Hardware Configuration

H. Software Configuration

I. Applications Configuration

J. Data Center Configuration & Specifications

K. Operations Schedules

L. Distribution List for the Manual

DISASTER RECOVERY BACKUP HARDWARE

- Keeping Your Company in Business -

VIPOOL M. PATEL

HEWLETT-PACKARD COMPANY

Abstract

As computers become more critical to the operations of a business, downtime becomes increasingly more costly. In some cases, it may even result in the shutdown of an entire business. Executives and MIS staff alike are seeking ways to protect themselves from the consequences of a system disaster due to fires, earthquakes, floods, sabotage or other reasons.

This paper deals with the need for having backup hardware. It discusses different alternatives for backup hardware, and addresses a common objection for not subscribing to a backup hardware service (Appendix A). Finally, the paper touches on practical issues of implementation such as developing a plan, working with a backup vendor, and rehearsing a recovery.

The Need for Backup Hardware

Backup hardware is a general term for the availability of computer hardware and peripherals for use in an emergency situation.

Backup hardware gives the MIS director more peace of mind, and it gives the executive more freedom from audit concerns. Although these are extremely important, the primary benefit of backup hardware is keeping a company in business.

There is a growing dependency on computer processing thoughout all areas of business. With this trend towards automation, a return to manual processing may not be achieveable in an emergency situation.

The following are examples of critical operations where the high cost of system downtime necessitates backup hardware.

Financial Services

Financial services manage the flow of money, a process that does not tolerate disruption. One HP customer interested in backup hardware is responsible for stock transfers and dividend payments. If a seventy-two hour turn around time is not achieved, a large fine is imposed by the Security and Exchange Commission. Most of the seventy-two hours are needed for data entry, printing, and distribution, leaving a very small window for the computer processing. In the event of a disaster, processing could not be delayed more than twenty-four hours.

Payroll

Payroll is a business basic that needs to be completed even if a disaster strikes. Timely payroll becomes even more important when there is tension existing between management and employees. One HP customer reported that union employees were ready to riot because paychecks were one hour late. Each day payroll is delayed could mean a day of lost employee productivity.

Orders

Sales order management is often sited as a critical application. Orders are a company's life blood and need to be quickly processed for continued operations. This is why HP gives the name "HEART" to its central ordering system. Orders from HP sales offices throughout the country are routed to HEART which redirects the information to appropriate divisions for product shipment. If HEART goes down, HP comes to a hault. This is one reason why HP has a fully rehearsed disaster recovery plan with backup hardware.

Distribution

Distribution operations are becoming very highly dependent on computing capability. One customer uses HP systems to make distribution decisions that send trucking assignments directly to the warehouses. Another company differentiates itself by offering twenty-four hour delivery through the use of computers. Delays in distribution results in customer dissatisfaction, higher inventories, and delayed invoicing.

Billing

Accounts Receiveable is an application which is common to most businesses. Businesses can not afford to provide goods or services to customers without billing in a timely manner. There are opportunity costs associated with the inability to track and effectively allocate revenues. Consider a phone company that records billing information and charges customers on a monthly basis. Each day those bills are delayed could result in lost interest on millions of dollars of revenue. The cost of backup hardware is insignificant in comparison.

Manufacturing

According to Infocorp, manufacturers comprise 45.8% of the HP 3000 installed base. Manufacturers use critical applications such as MM3000, HP Production Management and Work Order Control. These environments favor a local backup hardware solution because of the dependence on direct terminal connectivity.

Others

Almost any system, whether large or small, that is used to meet a deadline is a candidate for backup hardware. Newspapers can not make up missed issues caused by a system failure. Companies can lose large contracts because of a late response to a Request for Proposal.

Alternatives for Shared Backup Hardware

Although redundant systems, reciprocal agreements or service bureaus can backup computing capability, shared backup hardware may be more cost effective or reliable. Shared backup hardware means that one system is used to backup a fixed number of systems, usually operated on a first come, first served basis. The four most common forms of shared backup hardware are warmsite, hardware delivery, mobile and virtual machine.

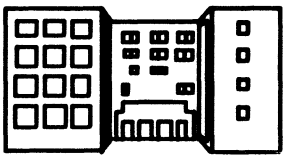
Warmsite

With the industry standard called a warmsite, the customer travels to a location which is fully equipped with a computer, peripherals, computer room, telecommunications, and office space. Because of all of these facilities and capabilities, a warmsite is more expensive than most of the other alternatives.

INDUSTRY STANDARD SHARED BACKUP HARDWARE

1) WARM SITE:

Customer travels to a fully configured system and computer room



2) HARDWARE DELIVERY:

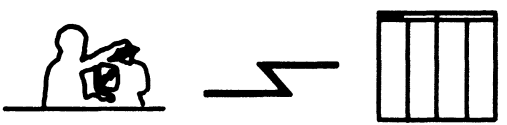
System is flown to a predetermined customer location within 24 hours

3) MOBILE:

System is delivered to the customer in an air-conditioned trailer with generator power

4) VIRTUAL MACHINE:

Customer uses terminals and data communications to connect to a remote computer



A warmsite is a good solution for systems in batch processing environments and for systems susceptible to disasters (such as flooding) when both backup hardware and computer room are required. The warmsite is ready to use as soon as the customer arrives, and rehearsals are conducted at the warmsite so the data processing staff knows what to do in the event of a disaster.

A well equipped warmsite will have leased lines and dial-up modems for businesses needing terminal support. Telecommunications requirements can make a warmsite an inappropriate solution, especially for customers who use communications lines that can not be redirected to the warmsite.

Hardware Delivery

Hardware Delivery means a computer and peripherals are flown or delivered to a pre-determined customer location within twenty-four hours. A computer room is not included, making this is one of the least expensive solutions. The customer needs to make separate arrangements for leasing a backup computer room if one is not available at the customer's location.

Hardware Delivery is a good fit for customers with systems from different vendors which all need to be backed-up at the same location. Since the computer is brought to a pre-determined location, existing or backup telecommunication lines can be used.

A rehearsal is not performed at the customer's location so the data processing staff does not get to practice in the same situation faced during an actual disaster and recovery.

Mobile

For customers who need a backup computer and facility but can not reroute telecommunications to a warmsite, mobile may be a good alternative. Mobile means a computer and peripherals are delivered in an air-conditioned trailer with generator power.

The mobile solution may require long travel time and is not effective in large cities where parking space is not available. Although an air-conditioned computer facility is provided, the customer still needs to supply office space and other facilities. Providing security in a trailer is also more difficult than providing security in a building.

Again, if a rehearsal is not conducted at the customer's location, it will not accurately represent the situation faced during an actual recovery.

Virtual Machine

The customer can use terminals and telecommunications to connect to a remote system. Users would continue to work at the same terminals or change locations if necessary. Processing can resume as soon as tapes are sent to the remote location.

This alternative reduces costs because the data processing staff does not need to travel for rehearsals or in the event of a disaster and because equipment is not moved.

Virtual machine is limited by the speed and availability of telecommunications. Creative solutions using satellite or other advanced technology continue to be investigated to provide more rapid and transparent recovery.

Implementation Issues

There are many practical issues to address when implementing backup hardware. Some of the larger considerations are developing a contingency plan, working with a backup hardware vendor and rehearsing a recovery.

Contingency Planning

In practice, backup hardware is often purchased before planning. However, it is extremely important to prepare in advance for the many operating and staffing procedures required for a successful recovery.

A contingency plan documents the procedures taken in the event of a disaster. In addition to documenting the steps needed to use backup hardware, this plan can include areas such as personnel safety, site relocation, responsibilities, insurance activities and notifications.

Choosing a Reliable Source for Backup Hardware

If a customer decides that backup hardware is necessary, it must come from a reliable source. Even though costs may be lower, protection decreases with backup systems that are used for other businesses or for timesharing.

Customers should be cautious of vendors offering backup hardware that can not be regularly handled. There have been instances of customers subscribing to backup hardware which did not exist.

Customers should also check the financial stability and history of a backup hardware vendor. The vendor should have the ability to obtain all needed support resources to contribute to a successful recovery. The vendor should also have the resources to upgrade so the customer's growth path is not limited.

Rehearsals

Rehearsals of recovery on a backup system have only proven the need for such a practice. Besides allowing the dataprocessing staff to practice the logistics, stepping through an emergency scenario can help avoid potential technical delays in recovering a system during an actual disaster.

During this type of dry run, one HP division encountered a problem with hardcoded passwords which could only be solved by recompiling the program.

Customers are more likely to have a successful recovery by preparing for a reload made during a rehearsal because it will be prepared for the configuration of the backup system.

The backup tapes must also contain an operating system which is suitable for the firmware in the backup system. Customers can encounter computing errors when the backup firmware is not compatible with their operating system.

Conclusion

Customers should understand the criticality of different functions of their business in determining their need for backup hardware. The recommended procedure is first to develop a plan, second to obtain backup hardware, and finally to frequently rehearse to ensure successful recovery.

Because of the time required to investigate their needs and alternatives, customers should begin the process as soon as they recognize their strong dependence on computer processing capabilities.

Vipool Patel is presently Systems Support Product Manager for the Product Support Division at Hewlett-Packard in Mountain View, California. In this capacity he is responsible for the development and marketing of backup hardware products. Vipool joined HP in 1984 and holds degrees in Industrial Engineering and Engineering Management and in Economics from Stanford University.

Appendix A - The Objection of Possible Multiple Disasters

A common objection to subscribing to a shared backup hardware service is the possibility of two customers needing backup at the same time. This is a strong objection for a customer concerned about wide scale disasters such as earthquakes, or power failures in an entire city. However, the objection becomes statistically weak if the concern is about two different customers having concurrent isolated disasters such as a fire.

Approximating the Probability of a Disaster

According to a 1985 study by International Data Corporation, in the U.S. there were 56,100 computer sites with a system valued over $100,000.

A conservative estimate of the number of disasters at those sites over that last five years is one hundred, averaging twenty per year.

Customers using a backup system because of a disaster are likely to have their computer replaced in one quarter (three months). Assume disasters in different quarters would not cause a problem of multiple demand. The quarterly average number of disasters is five.

The probability of a given site having a computer disaster during any one quarter is the average number of disasters per quarter divided by the number of sites.

$$X = \frac{\text{5 Disasters}}{\text{56,100 Sites}} = .0001 \text{ or } 1/10{,}000$$

The disaster rate for this sample can represent the disaster rate for the rest of the U.S.

Assumptions for Arguments 1 and 2:

- One hundred and one customers share the same backup hardware
- As shown above, the probability of customer Z having a disaster is X or 1/10,000
- The probability that at least one of 100 customers has a disaster is 100 times X
- For the purposes of isolated disasters, assume that the events are independent
- P(A and B) = P(A) x P(B) if A and B are independent events

Argument 1:

The first statistical argument says that since the probability of any customer having a disaster is so small, the probability of two disasters out of a select group of one hundred is insignificant:

The probability of Customer Z having a disaster and at least one of the other 100 remaining having a disaster is equal to the product of the independent probabilities which is 100 X^2 or 1/1,000,000.

It is argueable that the probability of one in one million is too insignificant to support an objection.

Argument 2:

The second statistical argument says that the probability that no one else is using the shared backup system at any time is close to 1 for 1.

The probability is 1 for 1 that no one else is using a backup system when it is a redundant system (not shared).

The probability that no one else out of one hundred others is using the shared backup system is 1 minus the probability that at least one customer has a disaster. This is equal to (1 - 100 X) or (1 - .01) = .99

By moving from a redundant system to a backup shared by 100 customers, the probability that no one else is using the backup system only decreases one percent. To some customers, this small sacrifice may be worth the costs saved with a shared backup.

An Approach to Networked Community Filing.

by: Andrew Pearce - Hewlett Packard
Office Productivity Division
Nine Mile Ride,
Wokingham, Berkshire
England.

Summary: The needs and working practices of groups of office workers are increasingly being taken into account in the development of office computer support products. Community filing systems must recognize and support the size and distribution of workgroups by providing for the organization and security of arbitrary groups of users. They must also have the underlying filing structures to assist in information retrieval by different workgroups.

HP File/Library, a recent product supplied as an option to HP DeskManager, uses the group membership feature of HP Desk to allow workgroups to be configured. It also has a simple "flat" structure of catalogs within a Library, rather than a more conventional hierarchical filing structure of folders within folders within drawers, etc.

This paper discusses these features with regard to how the workgroup management and catalog structure could be extended in the future to a networked community filing capability. In particular, how workgroup configuration and security control over networks can be provided.

The Rise and Fall of the PC workstation.

Today's office software products continue to strive for greater integration between different functions. In the HP Personal Productivity Center, electronic mail, word-processing, filing, time management, resource scheduling, graphics, spreadsheets, and database access, can all be accessed from one product interface. And the real action, we are told, is away from a terminal interface to a PC/workstation user interface.

Terms like "secretarial workstation", "desktop computer", "PC/workstation" have been around for some time now. All conveyed the vision that each office worker would have the power - and potential for productive work - of a full computer on their desk. Ask yourself in which areas the stand-alone PC/workstation has been most successful? Word-processing, spreadsheets, database management, and games are four certain little winners that spring to mind.

Not surprisingly, the most popular applications were those that supported stand-alone activities. Yet many real office tasks require users to switch from one application to another, and to share workloads. A tighter integration between functions was seen as one direction for improvement. Relatively overlooked up to now has been the need to support groups of people working independently on similar tasks, or parts of the same task.

Stand-alone wordprocessors were successful where an individual secretary or typist generated letters, memos, etc., or made changes to existing documents. Parts of much larger documents could be generated by several workers independently, and then brought together for publishing - but this required a great deal of (manual) organization. The stand-alone wordprocessor couldn't offer much help in this area.

The same is true of the spreadsheet. Several accountants given the task of composing a large spreadsheet could develop their own parts independently, but then had to organize sharing a master file into which they could build and link in their own sections.

Person-to-person communication has been supported by office mailing systems, and this has been one of the most familiar ways of sharing documents. (Another familiar way is swapping diskettes around). But mailing systems have primarily provided individual workers with the means of distributing their own documents to other individual workers. Until recently, mailing systems have not supported the activities of groups of users working together.

And what about filing all those documents that are produced and distributed? Most PC filing systems today provide individual workers with the means of managing their own set of documents.

The PC/workstation vision originated from the idea of providing support for personal organization, rather than providing support for a person in an organization.

Next bandwagon - Workgroup support.

Take the previous section as a light-hearted view of maybe why minicomputer-based software has withstood the PC office software onslaught - up to now. If there is to be a contest in the office between distributed PC LANs, and minicomputer hubs with central processes serving users on PC LANs, it will be won on the merits of software that can support groups of users working on joint activities.

A few people in the office and business systems industry have been trying to promote better computer support for groups of users for some time. Only now are we beginning to see support for workgroups appearing in office software.

HP's general office strategy is to move as much processing out to the PC/workstation as possible. The minicomputer will then be the central workhorse for application services, and will be particularly important for community document storage.

Given this approach, how can we best support groups of people working together? Three basic characteristics of workgroups to consider are:

A. Size. We need to be able to support workgroups of any size - from one person to everybody.

B. Spread. The distribution of workgroups is an important consideration. Some groups work together in the same room, others may consist of one or two workers from several departments spread around a building, others again consisting of individuals located on different bits of the planet.

C. Scope. "Scope" in the sense of the diversity of information and organizational support required for workgroup functions.

The third concept is not so straightforward, but is worth considering further. Doubtless, the tasks tackled by different groups of workers have quite different requirements for scope (or breadth) of information. Similarly, the workgroup activities that use the information are diverse.

Typically, a workgroup tackling a task with broad objectives will cast a wide net for information, and will need support for filtering and interrogating this information, as well as organizing the collective effort. An example is the group of consultants required to brainstorm a creative future business strategy for a company. It's a one-off task. They don't know where the information is, so they have to cast a wide net for information across public and internal networks. They need sophisticated information retrieval and filtering, and support for possible delegation of some tasks to sub-workgroups.

At the other end of the scale is the group of workers who use a known set of information in a well-defined and typically repetitive process. For example, a group of accountants compile a quarterly financial report. The information is obtained from well known sources, and the aim of the group is to speed up and automate the process as much as possible.

The flower in the storm - HP File/Library.

HP File/Library is a recent product introduced as an option within the HP DeskManager product. It provides community filing and archiving for users who are registered and can log onto HP Desk.

Users can file any documents they have in HP Desk in the Library. This allows them to share the documents with any other individual or group. When a document is placed in the Library, the user is prompted to supply fields for an electronic "index card". These fields are attributes that can be used later in specifying a search for the document. The user can supply four attributes:

Subject	: up to 60 characters
Author	: up to 60 characters
Keywords	: up to 240 characters
Comment	: up to 480 characters

The Subject and Author attributes are extracted automatically where possible.

The Keywords can be defined as a multi-value attribute; ie, delimiters are used to distinguish a number of separate keywords specified from one prompt. The other attributes are single-value, although individual words or substrings from their contents can be used when searching.

Four system-supplied attributes are set in addition to the user-supplied ones when a document is added to the Library. These are:

Document Type	:	HP WORD, MESSAGE, ADVANCEWRITE, etc.
Creator	:	The HP Desk registered name and location of the creator.
Date created	:	When the document was created.
Status	:	A document in the Library can be offline, checked-in, checked-out, pending archive, archived, or pending retrieve.

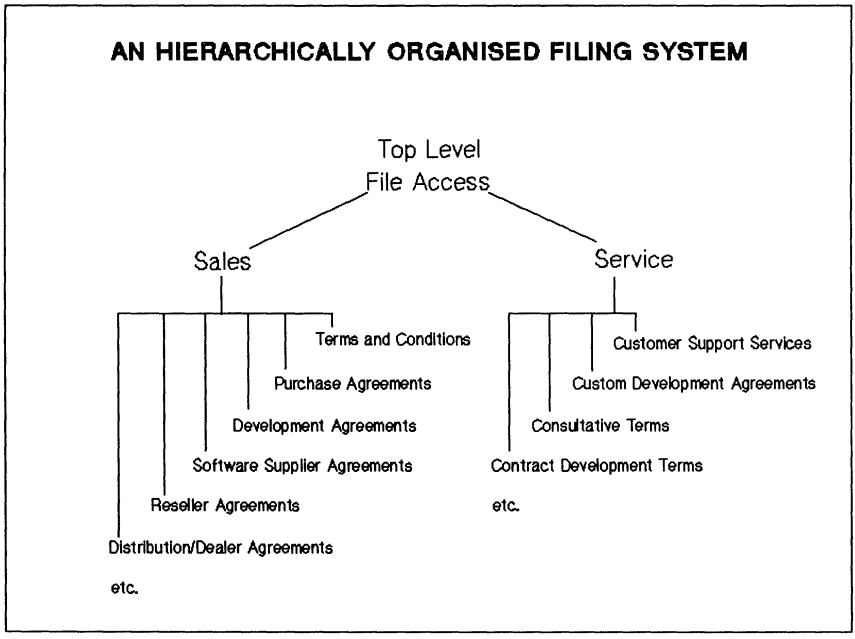
The user also specifies the catalog in which the document is to be placed. The Library is organized in three levels - with the Library at the top level containing catalogs at the next level, which in turn contain indexed pointers to documents at the third level. The documents could be HP Desk composite items, such as messages, but the parts within a composite item would not be independently indexed in the Library.

Why CATALOGS? Why not have cabinets, with drawers and folders?

Many people expect a filing system to have a deeper hierarchical structure. A filing cabinet containing drawers, which in turn contain folders, which may contain more folders and finally, documents. This expectation is not unreasonable given the existing HP Desk filing cabinet, and the paper office filing systems many of us are familiar with.

However, a simpler, flatter structure of Library, Catalogs, and Documents has certain advantages. First, there is no requirement for the user to know where to file a document - other than the appropriate catalog. For this reason catalogs can contain very large sets of documents - typically *all* the documents filed by a small department or workgroup - on any subject. They can also contain references to books and other documents stored offline.

In the hierarchically organized filing system, deciding where to put an item so that it can be easily found can be difficult and time-consuming unless it is well organized and all users agree on one set of filing rules. This may be appropriate for a specific application where a group of users simply want to replace an existing hierarchical filing system.



For example, this contracts department needs to keep two types of contract - Sales and Service. Within the sales section, items are are filed under sub-areas such as: Terms and Conditions, Purchase Agreements - common forms and several product lines, Demonstration/Development Agreements, Products Supplier Agreements, etc. A similar range of sub-areas could exist under the Service section.

A limited and known set of document types will be kept, and we can say that the "scope" of information to be retrieved is narrow and well defined. When a new type of document is added, a new place has to be found for it. Also, when the nature of the document is ambiguous, ie. its content spans several areas, then separate copies have to be filed in different places.

In HP File/Library we advocate replacing the area and sub-area names with keywords. So, when filing a Purchase Agreement, for example, the first keyword would be "Sales", the next "Purchase Agreement", the next "Widget Line #34", and so on.

This effectively increases the scope of information available for future retrieval. Suppose a group of users started using the same catalog for storing patent actions and proposals, some of which involve Widget product lines. Later, another group of users may need to search for all historical (archived) information on development and sales of Widget products.

The contracts group can only see the documents that they have added to the catalog. Likewise, the patents team only see their documents. But there is a set of documents about Widgets - some concerning contracts, others patents - that is now easily accessible to the third group.

This is basically the advantage gained by using a relational model over an hierarchical one. Originally unrelated, intersecting sets of information can be identified and retrieved. The bottom line for the user is that documents stored with one specific process in mind can be useful for future unanticipated requests by other users.

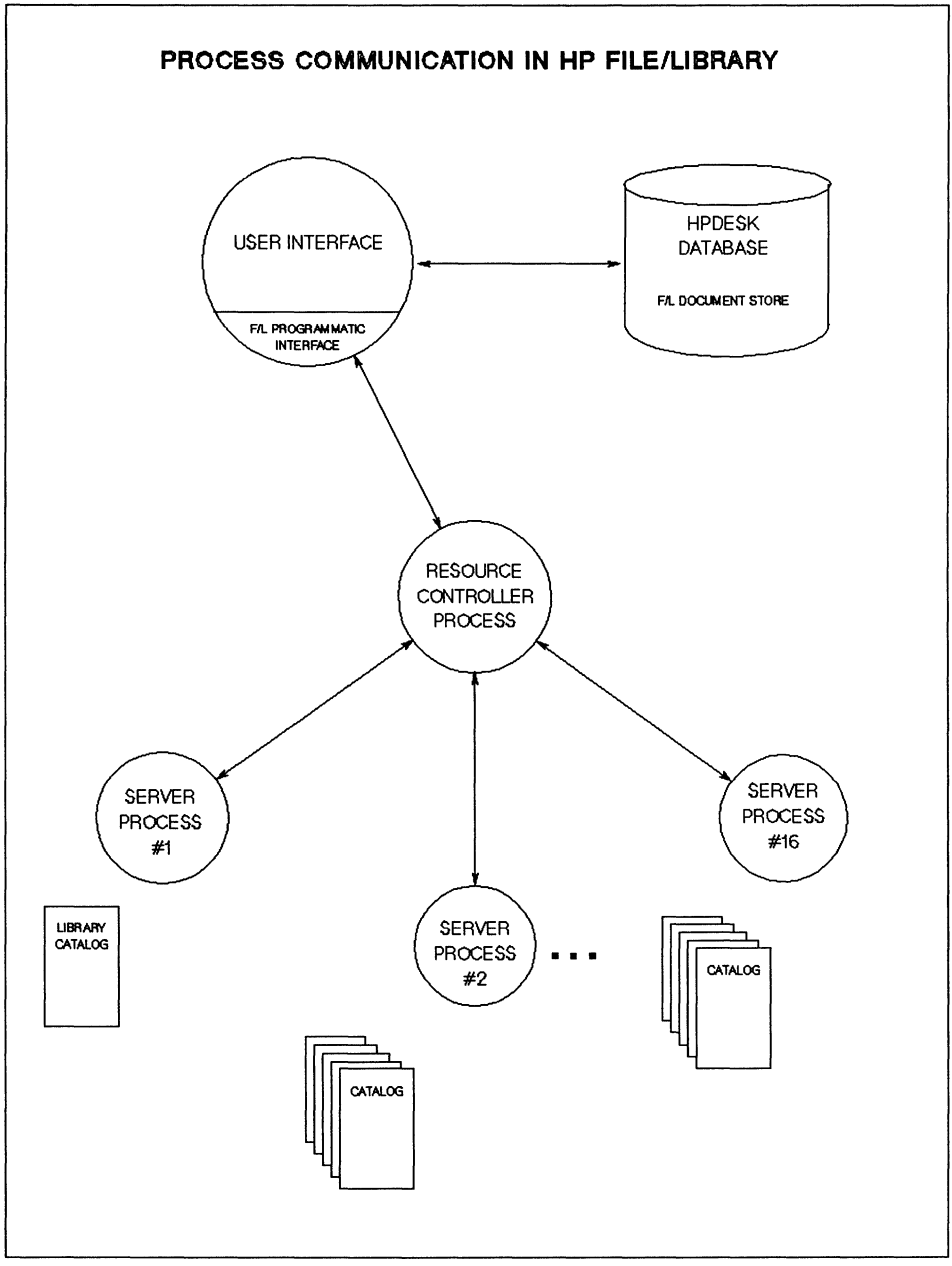
Another reason why we think the flat structure is good for community filing is simply the fact that all Libraries look the same. Anyone logging onto any Library on any machine knows and understands that structure.

Furthermore, when a user searches a catalog, the set of documents matched become his own personal view of the catalog which is persistent between his HP Desk sessions. The overlying structure of Library that he sees is unaltered, ie; the documents are still "in the same place".

An Approach to Networked Community Filing.

The key word of this section is "approach", because that's where we're at today. The first release of HP File/Library has been integrated into the HP Desk product as a new area. But the product is designed to be accessed by other programs.

The HP Desk user interface has been extended to provide commands within the Library and Catalog areas. This extension uses a programmatic interface to access the rest of the product. The HP Desk database is used for storing the content of documents that are indexed in the Library. Currently, any other program wanting to use Library would have to supply its own document store.


PROCESS COMMUNICATION IN HP FILE/LIBRARY
USER INTERFACE
F/L PROGRAMMATIC INTERFACE
HPDESK DATABASE
F/L DOCUMENT STORE
RESOURCE CONTROLLER PROCESS
SERVER PROCESS #1
SERVER PROCESS #2
SERVER PROCESS #16
LIBRARY CATALOG
CATALOG
CATALOG

When the Library is started, a Resource Controller (RC) process spawns a configured number of catalog server processes which are responsible for accessing the catalog files. One catalog server is normally dedicated to the Library catalog, and the other catalog servers are assigned catalogs as and when users request to open them.

Each user interface process logs onto the Library by passing a log-on request to the RC. From then on, the RC handles the passing of catalog access requests and replies between user interface and catalog server processes.

One major advantage of this design is that it obviates the need for catalog contention locking mechanisms. A catalog can only be assigned to one catalog server, and requests from different users to access the same entry in a catalog are simply handled in sequence in the order the requests are read from an interprocess communication (IPC) file.

A limitation of the current design is that it can only support community filing for users on one machine since every user accesses the Library by first logging onto HP Desk. Hence the workgroup size and spread considerations need attention in further developments of the product. Access to remote libraries may be achieved using LAN protocols and the HP Desk mail transport mechanisms. One important factor will be the recognition of classes of workgroups across networks of machines.

The Link between Security and Workgroup configuration.

Security has been provided at the three levels of the product:

Library	:	Manager access	- to configure access various groups have to the Library.
		Add access	- to allow catalogs to be added to the Library.
		Delete access	- to allow catalogs to be deleted from the Library.
		See access	- to see the Library.
Catalog	:	Manager access	- to configure access for groups and users to a catalog.
		Add access	- to allow documents to be added to a catalog.
		Delete access	- to allow documents to be deleted from a catalog.
		See access	- to see a catalog.
Document	:	Manager access	- to configure access for groups and users to a document.
		Edit access	- to edit a document.
		Copy access	- to copy a document.
		See access	- to see a document.

The philosophy of Library security is that any user, or group of users, only see those items for which they have been granted 'See' access. Thus, every user has potentially a different view of the catalogs available, and the documents listed within the catalogs.

Furthermore, there is no "super-manager" who can see all entries in the Library. All users can hide, or share, the documents for which they are responsible.

Groups of users are currently defined using the HP Desk group membership feature. This can be done by an HP Desk administrator using the configuration program. Once a group has been configured, specific types of access can be given to the group at any of the three levels of Library.

So, for example, a contracts manager asks his administrator to set up a group for his department. This is the group who regularly file away specific types of document into a catalog created by the Manager and called "Sales and Service Contracts". We can say that the scope of this information is "narrow", or well-known and defined. They are all granted 'See' access to the Library, and 'Add', 'Delete' and 'See' access to the catalog. The manager ensures that his group are granted 'Edit', 'Copy' and 'See' access to all documents as they are added by any members of the group and, in addition, gives another group called "WideAccess" 'Copy' and 'See' access to the documents.

Later, he agrees to rename the catalog to "Contracts and Patents" and gives a new patents group 'Add', 'Delete' and 'See' access to his catalog. The patents group can all edit, copy, and read their own documents, but cannot see any of the Contract documents. The "WideAccess" group is also given 'Copy' and 'See' access to all patents documents.

Much later, the contracts manager is asked to allow the WideAccess group to issue a search for information in the catalog, so he grants 'See' access to the WideAccess group. Members of the WideAccess group can then see (and copy) ALL the documents held in the catalog.

Clearly, the access needed by different groups is related to the scope of information they require to do their task.

The investigations for future possible releases of HP File/Library will consider how the current approach can be extended to enable Library access across networks of machines. The following improvements to our approach are under investigation:

- The method of configuring workgroups would be greatly improved by taking the responsibility away from the HP Desk administrator. There is also a case for specifying the scope of information required by a workgroup. This would enable a workgroup to inherit certain known and accepted rights for access to remote libraries.

- To maintain proper security, it is important that a broad scope group still asks to access catalogs that are normally hidden. This could be supported in software rather than as a personal request.

- The basic security mechanism is suitable for controlling access by any user or group from other locations. However, the maintenance of security could be improved by enabling access to be given to new groups retrospectively. Currently, this can only be done by changing the security for each item individually.

Concluding Remarks.

The central theme of this paper has been how well the current design of HP File/Library fits in with the growing need to provide workgroup support in a community filing application. We believe the three-tiered structure of a Library containing catalogs which contain lists of documents, has many advantages for community filing.

Three high-level characteristics of workgroups were discussed - Size, Spread, and Scope. We concluded that the current limitation of users access to one Library on one machine does not adequately provide for the expected spread of workgroups across a network. The scope characteristic was discussed at some length. Given additional facilities for workgroup configuration across networks, and some additions to the basic security mechanisms, the potential is there for storage and retrieval of broad and narrowly-defined sets of information by different groups across networks.

This gives us every reason to believe HP File/Library can be successfully extended to provide the network community filing capability for office applications in the future.

ANDREW PEARCE

Andrew Pearce is a software development engineer at HP's Office Productivity Division in Wokingham, England. During his three years with Hewlett-Packard he has been involved in developing and prototyping user interface software for HP 3000 and PC office applications.

References.

Leading Applications of Personal Computers.
"Office Views" Future Computing Inc. Third quarter 1984 survey.

"The Office Systems Cycle" by James H. Bair and Laura Mancuso.
1985; Hewlett-Packard Company

ABSTRACT

Solutions for Peripheral Sharing

John Peters, Hewlett Packard Company

Today's PC user needs access to sophisticated output capabilities in order to fully exploit the advanced capabilities of their PC workstation. Yet, peripheral costs make it impractical to provide each user with a full complement of high-capability output devices. In order to provide the advanced capabilities needed by office professionals and at the same time control costs, management must consider peripheral sharing solutions.

Management is faced with deciding between providing a low cost, low quality output solution or spending a fortune on a variety of sophisticated, high quality output capabilities for each user. Shared printing solutions can provide PC users with the printing capabilities they need and at the same time ensure that management is getting the best possible return on their peripheral investment.

This presentation will explore various shared printing solutions available and how management should decide which of these solutions best fits their situation. Specifically, the following areas will be addressed:

1. How should management evaluate its current and future printing needs?
2. What options are there for implementing shared printing? (switch boxes, printer buffers, dedicated printer servers, PC LANs, mini-based server LANs, other solutions)
3. How do the various options compare in terms of:

H P Vectra /Xenix

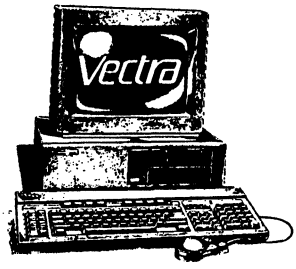
A Departmental Multi - User Machine

Gene Peterson
MicroAge of Scottsdale
13610 No. Scottsdale Road
Scottsdale, Az. 85254
(602) 483-9550

XENIX SYSTEM V

A MULTIUSER ANSWER FOR THE

HEWLETT PACKARD



Overview

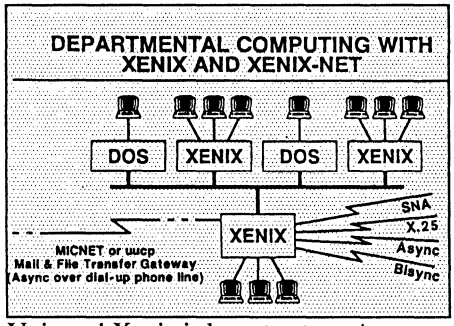
With the joint announcement of SCO Xenix on the H P Vectra by Hewlett Packard Company and Santa Cruz Operation, a whole new avenue of cost effective computing power now becomes available to the companies which have chosen to run their companies on the H P 3000 family. This joint venture brings together a very powerful operating system (Unix) on a very powerful computer (Vectra) and gives companies the opportunity to automate at a department level many operations which may have been to expensive or inappropriate because of the lack of software for the 3000.

This Operating System, as you will see as we continue is the cornerstone of a whole new approach to shared information and resource computing for Vectras.

This Unified environment where Xenix and DOS, Multiuser and LAN, and PC and H P 3000 systems serve each other and share resources -- each doing what it does best, without sacrificing any of their individual strengths.

Networking of Xenix and DOS systems through Xenix-Net. Micro-to-Mainframe communications through UniPath SNA-3270. Powerful applications such as Professional, Foxbase, and Foxbase + and Lyrix word processing are all powerful solutions that offers you the opportunity to create complete multiuser systems that span the range from Vectras to HP 3000 with more power, better price performance, and greater shared information and resource capabilities than ever before.

Applications for Xenix are readily available for almost every possible business entity. Word Processing with a shared processor (Vectra) is much more cost effective than a network due the cost of the stations which must be bootable computers of their own right, not to mention the LAN adaptors for the P C s, cabling and a fairly well trained and knowledgeable LAN supervisor. Other types of known personal productivity tools such as spreadsheets, data bases, compilers and communications to mainframes are readily available to the Xenix user.



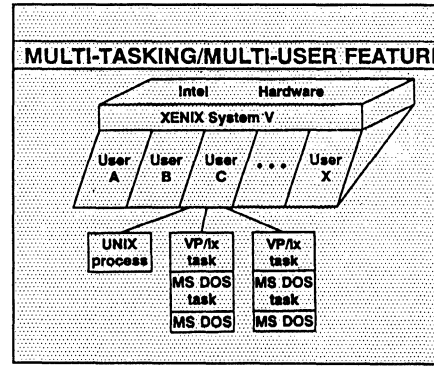
Unix and Xenix is here to stay. An enormous amount of people have made the Unix environment easy to work in and appropriate for the commercial applications it needed. Xenix by MicroSoft and SCO is a viable attribute for the H P 3000 computer department. It can only enhance the Distributed Data Processing Theme.

Xenix History

The early 80s found AT&T finally standardizing its venerabel Unix operating system. This was the finalization of at least 10 years of chaos and competing standards. Xenix is the most widely adapted version of UNIX, and presently installed on more than 85 percent of all microcomputers running UNIX, and on more than 50 percent of all UNIX machines. MicroSoft Corporation originated the Xenix operating system, but today, the Santa Cruz Operation sells it too. The two companies have a "Technology exchange agreement", where SCO is the vendor for Xenix on the 8086 and 80286 machines, and MicroSoft handles the large corporate deals, such as one that exists with IBM.

The primary reason for chosing System V over other versions of Unix is largely a matter of standardization. AT&T developed the System V Interface Definition (SVID), which specifies the hardware-independent interface between application and operating system. Xenix System V application developers who adhere to SVID functionally requirements can ensure that their application will run on any System V architecture, not just Xenix. This standardization makes programs and application portable to a wide class of machines, from P Cs to Minicomputers and to Mainframes.

The Xenix Operating System



The Basic System

Vectra/Xenix is a multitasking, multiuser operating system. On the Vectra this enables it to operate as a "Central Processor" with multiple terminals. The optimum number of terminals supported is eight. The terminals connected to the Vectra can be distributed among multiple users. Vectra/Xenix also provides for multitasking capabilities with allow the user to execute several tasks simultaneously. This substantially increases the user's productivity. Many other features are standard on the Xenix basic set and the following is a composite of these features and some explanation.

A XENIX GLOSSARY

adb: A general debugging command.
awk: A simple programming language that scans files for specified patterns.
Berkeley enhancements: A set of UNIX utilities developed at the University of California at Berkeley.
Bourne shell: A shell, or user interface, that is a UNIX standard.
csh (C shell): A shell whose style is reminiscent of the C programming language part of the Berkeley enhancements.
custom: A Xenix utility that allows you to selectively install or uninstall the major Xenix subsystems.
dbx: A high-level debugger found on Berkeley systems.
/dev: A directory that lists special device files.
eqn: A mathematics typesetting program.
kernel: The heart of a UNIX system. The shells and utilities request services from the kernel, which can talk to I/O devices, run programs, and manage memory.
ld: The lineage editor, similar to DOS's LINK command.
lex: A lexical analysis generator. Often used with yacc to develop programming languages.
lint: A C-program analyzer.
make: A program that automates the compilation of program source files.
man: A command that displays manual pages on the screen.
mkdev: A command that initializes a hard disk.
mkuser: A command that adds a user account to the system.
mm & ms: Macros used to control text formatting.
nroff: A text-formatting utility for standard printers.
pic: A utility that typesets line drawings.
refer: A utility that manages references for text processing.
SCCS: UNIX Source Code Control System, utilities for large-scale software development.
sdb: A high-level debugger used on System V.
shell: A program that causes other programs to be executed on command. For example, UNIX shells perform the very same task as DOS's COMMAND.COM.
tbl: A utility that formats tables.
troff: A text-formatting utility for typesetters or laser printers.
uucp: A utility that copies files to and from a remote system.
vi: A visual text editor, part of the Berkeley enhancements.
yacc: "Yet another compiler-compiler." A utility that converts context-free grammar into tables for parsing algorithms.

FILE AND RECORD LOCKING: File and record locking are critical to the multiuser environment. The need to monitor simultaneous accesses and updating of files such as databases. To maintain the integrity of data files, Xenix/Vectra provides record and file locking extensions that allow an application to lock either the file or the record.

MULTISCREEN FEATURE: A single keystroke switches the console screen and keyboard from one log-in session to another. With just one physical screen and keyboard, the user can chose up to 10 virtual terminals. Xenix's multitasking feature allows a user to concurrently execute 10 different programs.

Vectra can share software applications, data files, and peripherals in a common environment. Thus, the investment made in the purchase of a multiuser

PROGRAMMABLE FUNCTION KEYS: A "setkey" program allows support of over 50 programmable function keys on each virtual terminal. The keys can be

programed through simple escape sequences or the program.

CUSTOMIZED INSTALLATIONS: Xenix allows customized installations which can be single user and have the option of installing "Run Time Systems" using only 1.5 Mbytes on the hard disk as opposed to the 5 Mbytes normally required by a full flagged Unix operating system. Parts of Xenix can then be selectively installed or removed as desired.

LINK FACILITIES: These allow the user to add system supported devices to the system to support third-party hardware such as tape backup units. As these link facilities are incorporated into the base system, it eliminates the need to rebuild the entire Xenix operating system.

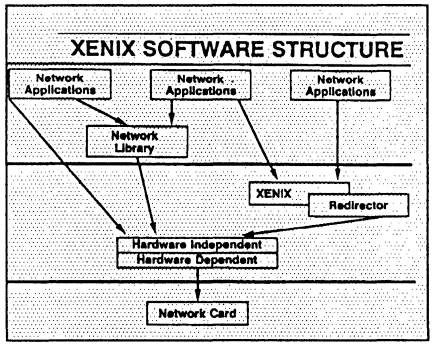
ADVANCED AUTOBOOT: SCO's Vectra/Xenix has an autoboot mechanism which automatically restarts the system after a power failure.

SUPPORT FOR ADDITIONAL SERIAL DEVICES: Add-on serial ports are supported for adding on ports beyond the standard com1 and com2 setup by DOS. These can be both cabled terminals and remote terminals over modem.

USER ACCOUNTS: User accounts help the Xenix system manager keep track of the people using the system, and control their access to the system's resources. Idealy, each user should have a user account. Each account has a unique "login name" and "password" with which the user enters the system, and a "home directory" where the user does his work.

SYSTEM SPOOLER: The system printer can be attached to a very effective spooler, which will handle all print requests from the users and que them up for printing. Remote printers are also supported, attached to other ports. Slave printers attached to terminal are also possible.

The Xenix Utilities



Many standard programs offer the user of Xenix the opportunity to manage the system, create new users, perform backups, add or delete peripherals devices and so on. Some of the more used and popular ones are defined and discussed in the following paragraphs.

GETTY: This is the main background program that supports the ports on the system and manages the communications. This includes the use of modems, what type of terminals and the intelligence of the terminal, baud rates and so on.

LPINIT: This is the main line printer management program. Initialization of the line printer is done through this utility, as well as the spooler.

CU and UUCP: These are the main utilities that are used to call remote systems and transfer data under Xenix. This programs dials a Hayes (or compatible) type modem internal or external.

LPADMIN: Many commands are available to the manage the line printers and the print jobs. LPSCHED manages the spooler control for ON and OFF. LPMOVE will reschedule all print jobs to another printer.

PS: This command tells what process status is on the system. Who the users are, what programs are running, what background tasks are operating and so on.

MAIL: A full mail system is always at access on the Xenix system. Messages can be sent to any user, stored, checked, deleted, printed, and read with options available in the mail utility program.

Xenix Languages

In the beginning, the Unix system was conceived by a few gifted programmers to meet their own needs. This latest version of Xenix doesn't abandon it roots: rather, it returns to them. Programmers benefit from Xenix rich array of tools and the fine tuning of many of the older tools.

APPLICATION DEVELOPMENT

- SCO FoxBase - dBase II workalike
- SCO FoxBase+ - dBase III Plus workalike
- SCO "SQL" Database - developer oriented
- Micro Focus VS COBOL
- Microsoft Languages

The single most important Xenix System V enhancement is the C Compiler. Most Unix operating systems are equipped a version of the Portable C Compiler. While it is portable, the Portable C Compiler has never been strong on performance. Xenix is equipped the Microsoft's acclaimed C-Merge Compiler. Most reviews of the C-Merge have noted its range of features and impressive performance.

- Despite miscellaneous shortcomings. Xenix System V is an easy system to administer.

Xenix Systems V C-Merge C compiler can produce programs running under DOS or under Xenix. Xenix executeable routines can be produced for System V or for Xenix III. Several styles of floating-point calculation and four memory models are supported, and the compiler can generate code for the 8086 as well as enhanced code that utilizes the instructions of the 80186 and the 80286. That is a host of features not found elsewhere.

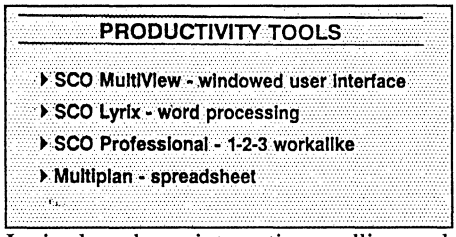
Fortran is part of most Unix systems but is extra cost on SCO Xenix. Pascal and MicroSoft Cobol is also available for a slight charge.

Off The Shelf Software

As compared to the Dos environment, Xenix is a little short on productivity software canned and ready to go. However, work-a-likes of the best are available so why look farther. They are well documented, run fast and handle

the job extremely well in the multi user environment.

Lyrix word processing is a comprehensive and fully configurable work processor. If you come from a Unix background, you'll find Lyrix an ideal system, combining a handsome and intuitive user interface with powerful features. If you come from a Dos background, however, you may find the lack of context-sensitive help and dynamic menus an incredible hindrance.



Lyrix does have interactive spelling and hyphenation, support for all terminals, and an easy to use interface to Unix/Xenix utilities such as electronic mail. In the multitasking, multiuser tradition, it offers file-locking and in the tradition of the UNIX, everything about Lyrix can be customized, including all commands and messages, special characters and foreign language, function keys, and printers.

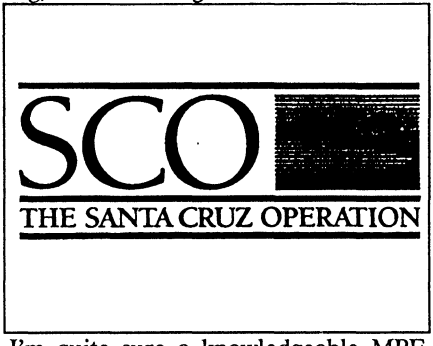
Professional works like the best known program of the DOS world, Lotus 123.

Other programs available are Informix Relational Database System, a good database which contains its own interactive query language and integrated data entry and maintenance program. Multiplan Electronic Spreadsheet, Microsoft Basic, Microsoft Fortran, and Unipath SNA-3270 are also available from SCO.

A comparison to H P 3000 M P E

MPE and Xenix do share a lot in common in design and layout. Both systems share concepts that make them great systems for multi user activity. While it would be unfair to say that Unix/Xenix is even close to the comprehensive nature of MPE, it could be fairly compared to MPE of the late 70s. Xenix 386 will address the integrated data base question with future revisions of FoxBase and SQL. C Language is emerging as the most compatible language for machine compatibility and is available. Implementations in MPE and Unix/Xenix give great use across machines.

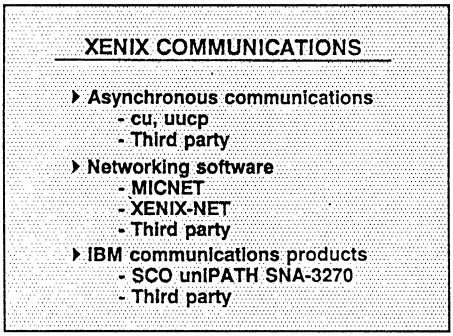
Generally, account, group and user access on Xenix is the same, the Public account set up of MPE is called Usr, the integrated spooler exists, file locking, record locking and such all exits.



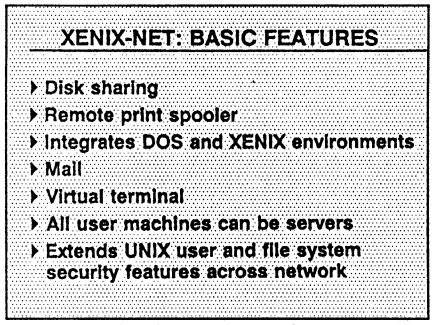
I'm quite sure a knowledgeable MPE System Manager could install a Unix/Xenix system on a Vectra in a day with one or two applications (packaged) up and running for users the next day.

Networking Vectra/Xenix to H P 3000 MPE

The Xenix file server produces full transparent access to MS-DOS and Xenix and can do the same to MPE. The server software consists of a number of kernel processes. A process is created for each remote access to the server. These processes are transitory and are terminated when the remote process is over. There is also a permanent listener process that manages the virtual circuits used by the network file system.



The Vectra Xenix system may perform concurrently as both a network file consumer and a network file server. The Xenix implementation is facilitated by the Xenix I/O structure. The Xenix file system presents a uniform interface for handling I/O objects. Every I/O object is represented as an inode in the file system where the type of the inode signifies how the I/O object is to be treated by the network. Accordingly, a network file is simply treated as an inode with type 'remote'. A remote inode is created when a remote file is referenced. A file is recognized as being remote when the path name of the file specifies that it exists on a remote device.



User applications access files through the system call interface. Since this interface is not modified by the network, application transparency for remote file access is assured. As all Xenix I/O objects reside in the file name space, and as most Xenix services are based on I/O objects that are now accessible over the network, many Xenix services will work across the network with little or no change. For example, Xenix mail can be used to exchange mail between Xenix users anywhere on the network as well as MPE sessions, and the at command can be user to initiate batch jobs on the other Xenix network systems.

Network file access is invoked when a system call references a remote file -- either with a remote path name or through an existing remote inode. The file consumer establishes the virtual circuit to the server system (if it does not already exist), initiates the remote file access transaction, and reports the file result back to the routine that originally invoked the system call.

On the server system, the "listener" process is responsible for establishing VCs to consuming machines. Once a VC is established, a request from a consumer process is serviced. This, in turn, activates a server process that ac-

cesses the local file system to complete the transaction. Thus every consumer process that accesses a network file has a "partner" server process to perform the work on its behalf in the server system. In effect, this is a natural extension of the manner in which commands are executed by the shell on a local Xenix system.

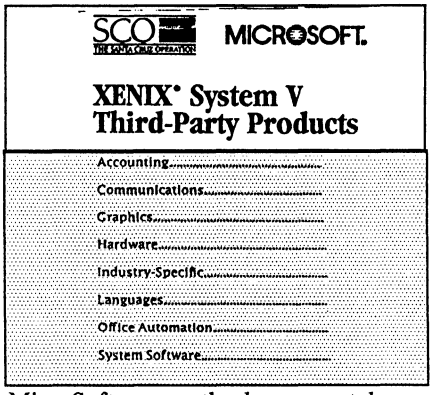
Xenix-Net is a separate function unit. There have been minimal modifications to the Xenix kernel. This facilitates the addition of Xenix-Net to an existing system.. That are numerous configurable options giving users the flexibility of tailoring Xenix-Net to meet their specific needs.

Xenix-Net provides the user with a device-level interface to the session layer. Thee user may create virtual circuits between applications anywhere in the network. These VCs can be used for any application-specific purpose. Typically, only layers 5 thru 7 reside in the Xenix Host system.

With all this flexibility, MPE designers can have all the file, peripheral and session access and integration they can use on a Vectra Xenix System.

Third Party Application Software Solutions

Application software is certainly not one of the drawbacks of Xenix today. Application software is available in literally every need from productivity types as in word processing and spreadsheets to the most demanding of vertical applications. Many catalogues exist covering the bulk of the software available. SCO's alone covers 353 pages of software available.



MicroSoft currently has a catalogue that covers 256 pages of applications available. Many more are available as more and more vendors convert their cobol, c and other language software to Unix/Xenix. Because of the flexibility of the operating system, it's easy to bring code from other machines to Xenix/Unix.

The Future of Xenix

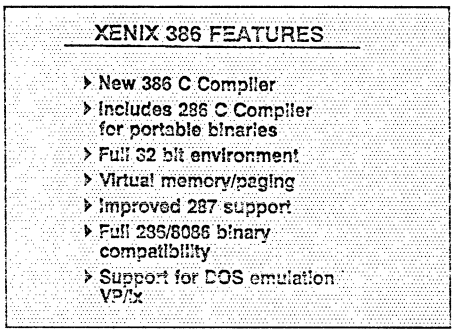
If the word UNIX makes you think of a university computer center full of hackers, its time to change your thinking. With their System V releases, both UNIX and Xenix are moving into the business world in a big way. With the advent of more and more application software ready off the shelf from third party vendors, UNIX/Xenix has to be considered in your plans for expansion or improvement in your current operating environment.

Because of the excellent quality of the operating system, and the utilities that exist, even new applications come to operation quicker, and are more used and accepted. Programmers and application specialists find that more flexibility exists than they can possibly

hope for, hence a favorable climate exists to get these new applications up and running.

UNIX/Xenix will be growing and expanding more and more in the near future. As this paper is being prepared, plans for the introduction of the 386 version of Xenix exist and should be announced by Interex in September. The power of a full 32 bit system with virtual memory and paging should give the system based on 386 technology power exceeding that of 16 bit minicomputers as we know them today. 386 Xenix will offer full 286 and 8086 binary compatibility and support DOS emulation.

80386 Processors



IBM DOS of the Future OS/2

With the introduction of the new architecture by IBM, the need for an operating system that would meet the needs of the users was certainly a necessity. OS/2 will meet these challenges but only for single user support. Use of megabytes of memory, multi-tasking, spooling just to name a few and more speed enhancements will all be the wave of the DOS future.

Multi-User does not fit in this environment however, except as a network of DOS machines. Connectivity is the word of IBM and will lead the product introductions into 1988.

Networks are a good file transferring device and central peripheral file management supervisor. Application programs which are really made to be central machine ordinate with multi-user access to the central files really work better, are more secure and fail less on a true multi- user operating system such as MPE or UNIX/Xenix.

The costs of a Network of several users with adaptors, cables, systems, and such usually is 25-35% higher than a comparable Xenix system for the same amount of users, and offers a higher degree of maintainability than does a Xenix system.

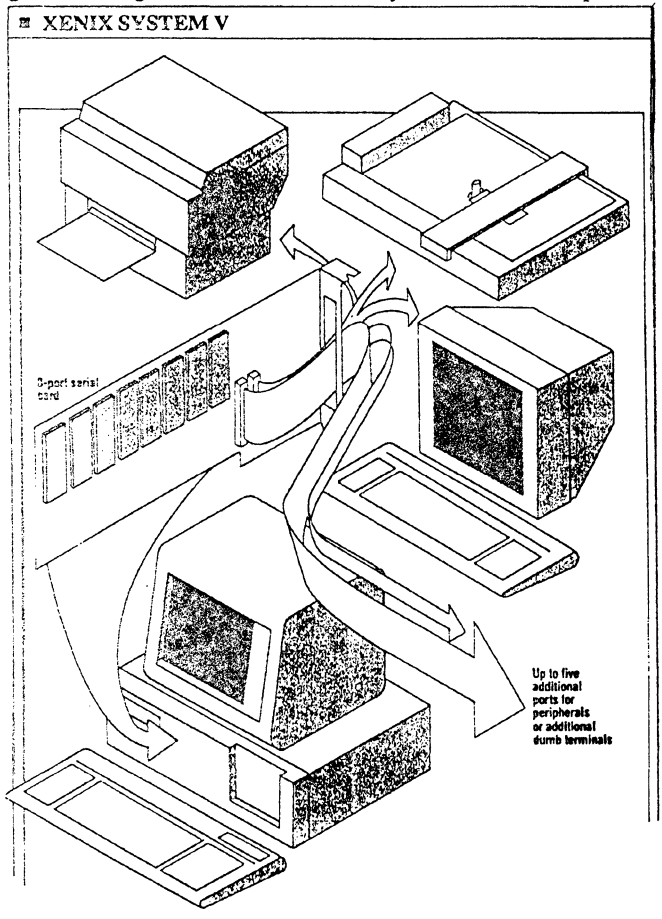
DOS is great for single user applications, and OS/2 will be even better, particularly for the "Power User" that needs the additional benefits which it offers.

Xenix in the future of Hewlett Packard

HP/UX is of course HP's implementation of Unix from AT&T and is an enhanced version that supports HP's proprietary chip sets on various machines including the new RISC technology currently shipping. Hewlett Packard has announced that it supports and the Xenix/Unix systems on the Intel Processors and will cooperate

with all concerned on future revisions. It appears that all the system manufacturers are jointly getting together on implementation of future revisions of the Unix Operating System. This will be great news for application developers and third party software companies. This will mean that migration of software between machines can evolve at even a greater pace than it has. One of the greatest longterm concerns of the system developer is the preservation of investment. This means that today's software and hardware efforts will continue to pay off for a long time to come without fear of incompatibility with future standards.

Manufacturers' technical concerns not only involve custom drivers for their products, but many other individualized aspects of that are being addressed by many of the Xenix developers such as SCO and Microsoft. Features supported, bug fixes, documentation are just a few of the challenges.



In conclusion, what this means generally is that an investment by a company in Departmental Expansion with Vectra/Xenix will not be faced with outdated applications and hardware next year. The Vectra/Xenix system will fit with the Distributed Data Processing Theme and Connectivity Theme of the years ahead and will be a low investment productivity enhancement to those small departments in need of these improvements and WILL give an honest return on your investment.

IMPLEMENTING AN APPLICATION TEST & MEASUREMENT FACILITY

Tim Brown and Patti Pribish
Hewlett Packard
2 Choke Cherry Road
Rockville, Maryland 20850

Abstract:

The authors describe how programmers and application designers can use available intrinsics to build an Application Test & Measurement Interface that will be extremely helpful in tuning an application's performance.

I. Introduction:

MPE provides the capabilities for programmers to gather elementary performance information. Calls to the intrinsics PROCTIME and CLOCK will return data about CPU utilization and elapsed wall time. By using these intrinsics, the capabilities provided by JCW's, and the PARM and INFO options to the RUN command, an application designer can implement a rudimentary measurement and test interface that will permit vital information to be gathered about an application's performance. Appropriate use of this facility will allow the designer to make controlled changes to the application and the system environment, and get more precise data about the effect of the change. This will allow scientific, measurable application tuning, rather than guesswork. This paper describes what information is available, how it can be accurately interpreted, and it demonstrates the usefulness of measuring changes. Sample PASCAL programs are included in the Appendix to document how to call and use the information.

II. The need for precise application performance measurement:

Over the years, a tremendous number of performance guidelines and tips have been published for use by the HP3000 user community. The authors have noted that students in the Designing and Optimizing Applications class and System Performance class are sometimes overwhelmed at the large number and wide variety of performance tuning options presented in these classes. Possible areas of investigation and tuning include:

- File placement on Disc
- CacheControl options
- Blocking Factors
- IMAGE buffspec settings
- Primary path maintenance (DBUNLOAD/LOAD or ADAGER DETPACK)
- Detecting & avoiding synonyms
- Use of "*" list construct
- Data Base redesign (eliminating unneeded paths, sorted chains)

- "Weak" locking strategies
  (lock/read/unlock/process/lock/reread/write/unlock)
- Large program with many segments (should we resegment)
- BUF vs NOBUF for file I/O
- Stack usage (MAXDATA, STACK, and ZSIZE)
  . . .

Considering the vast number of performance variables that an application designer/implementer can investigate, the authors have consistently recommended that available tools be used to identify the potential of most likely causes of performance problems. The most commonly used tools include those for making system wide observations, and gathering data base and application information. Some of the tools are the following:

System:
- OPT
- SURVEYOR
- TUNER

Consulting Services:
- HPTREND
- HPSNAPSHOT
- HPCAPLAN

Data Base:
- DBLOADNG or HOWMESSY
- DBSTAT2
- DICTDBA
- IDEA

Application:
- APS/3000 (Sampler)
- Proginfo

However, even with the available tools, the Data Base or application designer can still find himself in the position of guessing what is the most appropriate choice for his application, and having no application tools similar in function or power to the system measurement tools such as OPT. OPT, SAMPLER, IDEA, and TurboImage PROFILER all contribute data that will help the designer/DBA to make some performance decisions, and given enough time, the designer could perform a series of controlled tests and perhaps answer all his/her questions. But there is a need for an EASILY implemented test & measurement facility, so that the designer can make controlled changes and determine if performance has improved, worsened, or remained constant. Given the number of published recommendations on how to improve performance (some of which are contradictory), the time constraints under which the designer has to work, and the fact that applications and their environment tend to change over time, the need for a way to quickly measure and evaluate the effectiveness of any change becomes critical.

To quote from a now classic HP3000 paper, Jim May's "Programming for Performance":

> Intelligent attempts to improve performance require a disciplined plan of action. That plan must include at least four discrete items:
>
> -- Evaluation of existing performance
> -- Identification of candidates for alteration
> -- Selection and implementation of changes
> -- Evaluation of resultant performance
>
> [May 1-17]

May's paper, which we strongly recommend to any HP3000 programmer or designer, offers some excellent ideas and insights into how to improve program performance. He recommends the use of subroutines for several reasons, but one of the most compelling reasons he offers is that subprograms can be measured and tuned separately, thus allowing the programmer to accurately measure the results of the changes. Most importantly, the programmer can concentrate on the highest overhead items, and focus on those portions of the program that will provide the biggest performance return.

Unfortunately, many designers only execute the first three parts of May's action plan for improving performance. For instance, evaluation of the resulting performance gain after a Data Base UNLOAD/LOAD is rarely made. If a measurement was taken the question becomes, "was it taken properly?" One of the advantages to planning ahead for performance evaluation is that the measurement can be taken around a *logical transaction* which is specific to the application program. The users might have thought the response times were faster, but were they actually faster? Can an interactive user perceive a 10% improvement? Perhaps routine, repetitive batch job run times were compared, since that data is readily available, and some improvement, say 10%, was noted. However, the impact on the 65 online users was never objectively measured.

A possible solution to this problem would be the use of HPSNAPSHOT consulting. HPSNAPSHOT can provide very detailed information about average response times and average CPU seconds per transaction where response time is based on the completion of one transaction. The online transaction is the activity which occurs between the satisfaction of a terminal read and the point at which the next read is issued. Some terminal reads are not actually realized to the user as a completed transaction (for example a terminal status read for control purposes. However, there are a great many independent variables that an application designer/programmer may want to manipulate, as noted above, and these variations may be over an extended period of time. The HPSNAPSHOT consulting product is not designed or intended to be re-run after every change to the Data Base design, or blocking structure, etc., although multiple data collections for HPSNAPSHOT are an option to the product. The authors recommend users discuss the appropriate use of HPSNAPSHOT with their HP representative.

The intent of a user implemented Application Test & Measurement Interface that the authors are recommending is to capture readily available performance data with a relatively small investment of user time and effort. The authors also recommend a data management strategy for this performance data that will assist the designer in easily evaluating the consequences of his tuning efforts, and to establish a performance "base line" for an application. This base line will help a designer/DBA person to quickly identify performance degradation within an application, and take appropriate action.

III. Information about Implementation

In it's most basic form, a user test interface could be implemented by capturing start and stop times for user transactions. This information is easily supplied by MPE by calling the CLOCK intrinsic. Also, CPU utilization is readily available by calling PROCTIME. In a simple application where the user enters some screen data, presses enter and waits for a some data base retrieval or update, the programmer would bracket the logical transaction with code to collect the performance data. The procedure would look something like this:

- VREADFIELDS completes
- call CLOCK and save current time in START-TIME
- call PROCTIME and save current in START-CPU
- do some work: get a few records
- update some records
- call CLOCK and save current time in END-TIME
- call PROCTIME and save current CPU utilization in END-CPU.
- write an application event record to a file with
  event-id, user-id, start and end values.

Essentially, to gather timing data from MPE would formalize what many end users are already doing, and of itself is not much more helpful than using a stopwatch to time the system response. However, by gathering many timing samples from many terminals over a reasonable period of time, it is possible to get more statistically meaningful data. Also, trends over time, (or better yet, after changes to blocking factors and the number of paths in a detail), could be more accurately measured. And if CPU usage is collected with PROCTIME, then the serious evaluation of tuning attempts could be done.

Besides just collecting this raw timing data, the implementor should plan to do post-processing of the data. Writing to an MPE flat file makes sense to keep the interface simple and limit interference with measurement of the application. However, it would make sense to load the data into an IMAGE Data Base, and use existing Data Management tools to calculate average transaction event wall time and cpu time. Also, data that is out of reasonable bounds should be identified and eliminated from the sample. (If a data base update ordinarily takes 3 to 6 seconds, and one collection shows 83 seconds elapsed, but the CPU seconds was within range, we have to consider whether we want to use that record in the evaluation of the blocking factor change. The user

hitting the break key, a PURGEACCT, or many other system perturbations could have been responsible for the peculiar result. A data point such as this in a small sample could completely warp the results.

Also, it makes sense to set up the program so that the data collection code is activated conditionally on an INFO or PARM statement, or JCW value. This way it can be switched on or off at run time. This allows greater flexibility in deciding when to use, or implement this elementary performance measurement interface.

IV. Words of Caution

There are two major types of measurement on a computer system: hardware and software measurements. The form of measurement discussed in this paper is a software measurement. The nature of software measurement makes certain words of caution very important. A software measurement uses additional system resources to those being used by the application program so the resulting time includes the time to execute the application code as well as the measurement. One of the key resources to be considered is the Operating System. MPE is responsible for deciding which process obtains the CPU through the DISPATCHER/SCHEDULER, this decision is based on a priority scheme. One of the most difficult parts of a good measurement is testing in a controlled environment. An application program running on a system with high level of data communications activity, in general, take slightly longer than it would on the same system with the data communications disabled, since data communications runs at a higher priority than a default user process. There are a great number of other examples were the DISPATCHER will make a decision that will change the resulting test data. To help offset the effects of the environment here are some of the things to consider:

* The measurement code should be placed within the same code segment when possible to avoid excess segment transfers.

* To make a valid comparison before and after an application modification, the testing environment should be consistent.

* A base line for an application timing should be done for each version of the operating system and not carried over from a previous version if the measurement goal is to look at the effect of changes that have been made within the application program.

* Several measurements should be done and averaged to help minimize the effect of the changes in the environment.

* Any manipulation of the timing records should be done AFTER the test has been completed.

* A very short program will have a higher degree of deviation between measurements, and it will be difficult to see a

significant change in the application timings to the point of being invalid.

* As noted above, data values that seem unreasonable should be discarded from the sample.

V. Definition of a user test & measurement interface:

The user should document the environment in which the test was taken, the changes that were made, and the expected results. The environment can be documented by including the version of the operating system as well as information about the current utilization on the system. The changes should be clearly documented so that the test can be duplicated if later testing is needed. The expected results should be documented in terms of what time from the measurements are expected to decrease. Over time a rough guess as to the percent of change expected may be possible.

Interface: This is the user procedure called at program startup, exit, and any number of times in between (possibly around a logical transaction) that returns CPU seconds consumed and elapsed wall time between any two events. We recommend that the intrinsics be called from inline code to avoid additional overhead of segment transfers. The best way to collect this data is to write the timing information to an MPE flat file, and do the calculations or posting to a data base later. Many may feel that after writing the data to a Data Base, more statistical analysis should be performed, over and above than just calculating averages. However, the information that can be gathered from even simple collections and analysis can be invaluable in identifying the important performance variables.

VI. Example of appropriate use:

An example of appropriate user test and measurement in the Pascal programs can be found in the Appendix, part A, examples 1 and 2, and will be referred to in this section. The program being tested is a prime number generator and has the default Pascal compiler option $RANGE ON. The change made in the second program is that the $RANGE OFF compiler option was used. The $RANGE ON (the default) causes the Pascal compiler to generate range checking code for assignments, array indexing, etc. as discussed in the Pascal Reference Manual (part-no 32106-90001). It was rumored that turning $RANGE OFF could significantly improve Pascal run-time performance.

The test was done on the same version of the operating system within a 5 minute time span. During this interval the :SHOWJOB and :SHOWQ indicated no other significant changes in the operating environment. The values given in the Appendix A indicate one example run of the test. An example of a possible way to document the environment for this simple test can be found in Appendix B.

Notice that the results indicated that for this mathematical Pascal Application turning $RANGE OFF caused a 55% decrease in the CPU seconds and a 74% decrease in the WALL TIME. This data will change in a different environments, and in this case a CPU bound system should show an even greater improvement with the $RANGE OFF. So as this implies knowing the environment in which the application is running is very important when determining the actual change that is realized.

VII. Conclusions:

The need to measure application performance while experimenting with system and application variables is essential. The authors have delineated how users can exploit existing intrinsics on the 3000 to capture basic performance information. Limitations of this method have been explained, and users should be aware that, as with most tools, the data can be easily misinterpreted. We have suggested how the raw data may best be formatted and gathered into a appropriate data structure (IMAGE Data Base) to make it manageable so that reports and graphs might be produced. Finally, the authors have included an example of calling the PROCTIME and CLOCK INTRINSIC.

VIII. Appendix

A. Pascal prime number example
<< simple pascal program to calculate prime numbers included
as an example . . . >>

1. The first program shows the Pascal prime number program with the compiler option $RANGE ON.

```
:PASCAL
primels.pascal

program primel (input, output);  {Mar '85 Interact p. 80}
                                 {Stan Sieler }
                       {ADDITIONAL CODE IN UPPER CASE - jtb}
{ PRIMEl - USES DEFAULT COMPILER OPTIONS }
{$RANGE OFF$} {RANGE CHECK ON BY DEFAULT}

TYPE
   TIME_TYPE  = PACKED RECORD
                  CASE BOOLEAN OF
                    FALSE : (ALL : INTEGER);
                    TRUE  : (HOUR, MINUTE, SECOND, TENTHS :
                                0..255);
                  END;
const
   loops =   10;
   size  = 8190;

procedure    primes    (loops    :
integer);

var
   flags : array [0..size] of boolean;
   i, prime, k, count, inter : integer;
   CPUSECS : INTEGER;
   WALLTIME, WALLTIME_START : TIME_TYPE;

FUNCTION PROCTIME : INTEGER;INTRINSIC;
FUNCTION CLOCK    : INTEGER;INTRINSIC;

begin

WALLTIME_START.ALL := CLOCK;
writeln ('Begin PRIMEl ($range on$) ',loops:l,' iterations',
         chr(7));
for inter := l to loops do
   begin
   for i := 0 to size do
      flags[i] := true;
   count := 0;
```

```
   for i := 0 to size do
      if flags[i] then
         begin
         prime := i + i + 3;
         k := i + prime;
         while k <= size do
            begin
            flags[k] := false;
            k := k + prime;
            end;
         count := count + 1;
         end;
   end;
 writeln (chr(7), 'done, #primes = ', count:1);

 CPUSECS := PROCTIME;
 WALLTIME.ALL := CLOCK;
 WALLTIME.ALL := WALLTIME.ALL - WALLTIME_START.ALL;
 WRITELN('PROCTIME',CPUSECS);
 WRITELN('Walltime:'  ,WALLTIME.HOUR    :1,
                  ':',WALLTIME.MINUTE  :1,


                  ':',WALLTIME.SECOND  :1,
                  '.' ,WALLTIME.TENTHS  :1);

end;

begin
   primes (loops);
end.

:RUN primel



Begin PRIMEl ($range on$) 10 iterations*
*done, #primes = 1899
PROCTIME        4439
Walltime:0:0:8.8

END OF PROGRAM
```

2. The second example shows the Pascal prime number program with the compiler option $RANGE OFF.

```
:PASCAL prime2s.pascal

  program prime2 (input, output);  {Mar '85 Interact p. 80}
                                   {Stan Sieler }
                        {ADDITIONAL CODE IN UPPER CASE - jtb}
  { PRIME2 - OVERRIDES DEFAULT COMPILER OPTIONS }
   $RANGE OFF$  {RANGE CHECK ON BY DEFAULT}

  TYPE
     TIME_TYPE  = PACKED RECORD
                    CASE BOOLEAN OF
                      FALSE : (ALL : INTEGER);
                      TRUE  : (HOUR, MINUTE, SECOND, TENTHS :
                               0..255);
                    END;
  const
     loops =   10;
     size  = 8190;

  procedure    primes    (loops    :
  integer);

  var
     flags : array [0..size] of boolean;
     i, prime, k, count, inter : integer;
     CPUSECS : INTEGER;
     WALLTIME, WALLTIME_START : TIME_TYPE;

  FUNCTION PROCTIME : INTEGER;INTRINSIC;
  FUNCTION CLOCK    : INTEGER;INTRINSIC;

begin

  WALLTIME_START.ALL := CLOCK;
  writeln ('Begin PRIME2 ($range OFF$) ',loops:1,
           ' iterations', chr(7));
  for inter := 1 to loops do
     begin
     for i := 0 to size do
        flags[i] := true;
     count := 0;
     for i := 0 to size do
        if flags[i] then
           begin
           prime := i + i + 3;
           k := i + prime;
           while k <= size do
              begin
```

```
            flags[k] := false;
            k := k + prime;
            end;
         count := count + 1;
         end;
    end;
  writeln (chr(7), 'done, #primes = ', count:1);

   CPUSECS := PROCTIME;
   WALLTIME.ALL := CLOCK;
   WALLTIME.ALL := WALLTIME.ALL - WALLTIME_START.ALL;
   WRITELN('PROCTIME',CPUSECS); {I:3?}
   WRITELN('Walltime:'  ,WALLTIME.HOUR    :1,
                    ':',WALLTIME.MINUTE  :1,

                    ':',WALLTIME.SECOND  :1,
                    '.' ,WALLTIME.TENTHS  :1);

end;

  begin
    primes (loops);
  end.

:RUN prime2

Begin PRIME2 ($range OFF$) 10 iterations*
*done, #primes = 1899
PROCTIME        2046
Walltime:0:0:2.255

END OF PROGRAM
```

B. Here is an example job stream that allows you to document the environment.

```
!Job perftest,user.acct
!comment A SHOWME is done to document the current
!comment operating system
!showme
!comment The showjob command will give the job/user logon
!comment information
!showjob
!comment
!comment    - a showcache will document whether caching
!comment    is enabled
!showcache
!comment The showq information will give information
!comment regardingthe DISPATCHER queue and the number of
!comment processes in the various queues.  For an
!comment of :SHOWQ information see the :TUNE command
!showq
!comment Now run the job with the measurement enabled
!run PRIME1
!EOJ
```

THE END USER MODULE OF HP'S BUSINESS OFFICE SOLUTION

ORMOND RANKIN
HEWLETT-PACKARD
FT. COLLINS, COLORADO

INTRODUCTION

Managers, professionals and office workers within a business office environment have a variety of networking needs; by meeting these needs, businesses can improve communications, cut costs and boost productivity.

Common end-user needs include convenient access to: productivity tools, host and PC server resources, host applications from a PC or terminal, communications and sharing of information among workgroup, department and site-wide workers.

This paper reviews the options available for meeting these needs with Hewlett-Packard's end user capabilities within the company's Business Office networking solution. It covers end-user connectivity alternatives for individuals, workgroups, departments and entire sites, while focusing on both PC and terminal connectivity using LAN, remote asynchronous, data switch and PBX technologies.

Over the last several years the demand for integrated office automation solutions has grown dramatically among business users. Hewlett-Packard's HP AdvanceNet Business Office Network solution offers a variety of flexible, integrated approaches to meeting communications needs within the business environment.

A key issue facing end users in the business office is PC versus terminal networking. The issue is really one of cost versus functionality and power. Terminals offer a display screen as a window into the multiprocessing system they are connected to. On the other hand, PCs add inexpensive and constant local computing power with the capability of offloading multiuser systems.

The difference in cost between a terminal and a PC can range from less than $500 at the low end to upwards of several thousand dollars at the high end. Networking to support the PCs provides greater functionality and is correspondingly higher priced. But the investment provides low-cost yet powerful productivity tools for the office worker, and an extension of the multiuser capabilities to a dedicated low-cost third tier of computing power; in other words, the extension of distributed data processing to the workstation level.

Whether they choose PC or terminal networking, end users within a business office need to meet their cost and application needs. HP offers choices for such users that lend themselves to the needs of different users. These product alternatives are organized by the different ways end users commonly develop networking solutions.

The major alternatives are:

--Individual Connectivity-- This includes connecting workstations to HP 3000s where individuals are in either remote locations or in areas where they have the sole workstation (terminal or PC).

--Workgroup Connectivity-- This covers networking clusters of terminals to an HP 3000, and networking groups of PCs in a local area network with a PC server.

--Departmental Connectivity-- This is for networking larger groups of PCs and terminals to an HP 3000 within a department, and for networking to other HP 3000s on a backbone network or in a computer center.

INDIVIDUAL CONNECTIVITY

Individuals require productivity tools such as word processing, graphics and spreadsheets as well as access to mail and database facilities. In addition, the ability to access computer center information and resources (e.g., discs and printers) may be of value.

Terminals with departmental and computer center processors can provide these capabilities with RS-232 connections and applications such as HPWord, HPDraw, Deskmanager and Query with the Image database. Using Network Services/3000, a terminal user can obtain information from other networked HP 3000s. PCs provide the same capabilities, but can perform many local needs with applications such as Graphics Gallery, Executive MemoMaker and/or Advancewrite, Lotus 1-2-3 and AdvanceMail.

These terminals, PCs and printers are connected to the HP 3000 through the Distributed Terminal Controller (DTC) for the new 900 Series of HP 3000s, and through the Advanced Terminal Processor (ATP) for all other HP 3000s. These interfaces support both RS-232 and RS-422 with data rates up to 19.2 Kbps.

Users can individually access host-based applications such as electronic mail, accounting and financial packages. In addition, some users might direct output to a local printer.

Local or remote PCs are attached by a built-in RS-232 interface, and they require HP AdvanceLink terminal emulator software to give terminals the capabilities mentioned above, plus file transfer capability.

For remote PCs, end users may want to add HP Serial Network software for the PC, and HP Asynchronous Serial Network Link software on the HP 3000 to provide access to HP's Personal Productivity Center (PPC) software.

Adding such capabilities as Resource Sharing gives the ability to use HP 3000 discs and printers as if they wree directly attached to the PC, as well as the ability to backup PC discs directly to the 3000. Choosing this option means greater integration for the PC and improves ease of use and transparency for the user.

These features of the individual connectivity alternative provide numerous concrete benefits. Shared applications and information can be centrally located to facilitate sharing and lower

costs. By transferring files between the HP 3000 and a PC, data is available where it's needed without re-entry.

In addition, transparent access to a host from a remote PC means that people don't have to be at the office to access whatever system resources they need.

WORKGROUP CONNECTIVITY

Workgroups of terminal users can be linked in the same way individual end users are connected, or through multiplexers. This gives groups of terminal users local access to host-based applications, as well as the ability to print files on an HP 3000 printer or local serial printers.

A Local Area Network (LAN) connection is more efficient for groups of PC users. Such users can share discs, files, printers and spooling capabilities, as well as plotter sharing and spooling on a PC server on a high-speed LAN. These users can also have direct access to HP 3000s in a data center for applications such as HP DeskManager (electronic mail) and Information Access for access to corporate databases.

A workgroup LAN is built by connecting PCs in the workgroup to a StarLAN Hub. HP StarLAN is the company's twisted-pair LAN that fully integrates PCs and HP 3000s. Twisted-pair wiring offers flexibility, efficiency and low-cost wiring both during installation and over the long term.

In addition, HP 3000s in a data center can be accessed directly from the workgroup PCC LAN if a StarLAN Bridge is used. This bridge connects the StarLAN Hub to an IEEE 802.3 thick or thin coaxial cable.

The workgroup connectivity alternative offers all the features described in the individual connectivity alternative, plus the capabilities for PC users to:

--share discs and files from a PC server;

--share a PC printer or plotter attached to a PC server;

--utilize high-speed links;

--run popular standalone PC applications as well as most software that supports Microsoft-Networks (R) MS-NET (R) applications over the PC LAN; and the ability to run the entire HP Personal Productivity Series of major applications such as Lotus 1-2-3, Executive MemoMaker, Information Access, Resource Sharing and AdvanceMail over a high-speed link;

--access corporate databases and HP 3000 applications;

--use twisted-pair wiring.

DEPARTMENTAL CONNECTIVITY

This alternative provides all the computing resources and OA capability needed by a large department of end users. It allows connection of an HP 3000 located either within the department or in the data center to terminals, printers and PCs. It also enables the entire department to connect its site backbone to access additional information and resources located in the data center, or in other departments and workgroups along the whole network.

The HP 3000 is the heart of departmental connectivity alternative. Terminals are linked to it in the same way as in the earlier alternatives, and workgroups of PCs are connected to the HP 3000 via a LAN.

PCs users access the resources of the HP 3000 transparently by running Resource Sharing software on the HP 3000 and StarLAN user link software on the PC. Users can also choose and format menu commands transparently without knowing the database structure, and download them into popular PC applications such as Lotus 1-2-3, dBase II, and R:Base 5000. This requires Information Access software on the HP 3000 and StarLAN software on the PC.

And finally, all the applications of the HP 3000 normally accessible from a terminal are still available through terminal emulation over the network.

These capabilities add up to a high degree of PC-terminal-host integration. User access to systems across departmental boundaries and subnets is easy. Finally, where low-speed, lower functionality connections are acceptable, users are able to switch terminals and PCs by means of data switches and PBXs.

While each of the three basic alternatives within the End User Module is addressed to different environments, it's also possible to combine elements of the different options to create connectivity solutions within offices.

CORPORATE-WIDE CONNECTIVITY

End users have several alternatives for communications on a corporate-wide basis. One alternative provides gateway services for end users at a local site. An HP 3000 can be used as either a dedicated or shared communications gateway. An HP 3000 can also be used as a LAN to X.25 gateway for other HP 3000 systems as required. If datacomm traffic is high, users may want to dedicate the system to the gateway function.

This gateway offers users several important features: interactive and batch data exchange, single X.25 network access for all LAN-based systems, and use of the same access link for remote workstations and systems. These features minimize network access costs significantly.

If users require access to an SNA company-wide network, a second alternative is an SNA gateway. This is the best solution for PCs on the LAN and terminals connected to 3000s. For higher datacomm traffic, multiple gateways may be needed.

This gateway provides SNA batch job submission, SNA interactive access to 3270 applications, and access to DISOSS (via SNA LU 6.2) or PROFS for electronic mail exchange with HPDesk users.

ABSTRACT

Terminal Servers Update

Alic Rakhmanoff, Hewlett Packard Company

Terminal servers provide a cost-effective and flexible way to logically connect terminals and printers to hosts in a local area network. These terminal controllers are used in both the business office environment and the manufacturing environment, however, specific use and configuations vary in the two different environments.

This presentation will cover HP's terminal controller strategy in the business office environment and the manufacturing environment with the emphasis on the business office for this commerical Interex conference. The following topics will be addressed: funtionality of HP's Distributed Terminal Controller (DTC); connectivity via baseband and broadband LANs; connectivity to HP Precision Architecture HP systems and to non-HP systems; recommended configurations; future enhancements to today's solution as well as a migration strategy.

System Performance Measurement

by Ron Reimert, Unison Software, Inc.

This presentation is a management level review which will analyze common techniques of measuring system performance. It is intended to stimulate better systems decisions based on more accurate analysis of relevant, objective data. Our goal is to relate performance to measurable MIS goals and not to vague, elusive technical terms. Issues will be focused around the world of the customer, the user in our company, and their ability to contribute effectively to company profits.

When a company makes its first acquisition of a system, they usually select it based on objective measurements. The hardware and software are selected to solve a specific management problem. Whether it's HP or another manufacturer, we systematically analyze our situation. We determine first what our problem is and what we must do to correct it.

We try to select a wide range of objectives and needs to help solve our problems. We analyze various hardware and software alternatives against the needs and hopefully optimize our solution. What is a good solution? It is usually defined as user friendly, efficient (easy to use or install), cost effective (price performance ratios or person hours saved or errors reduced) and of course we make better decisions. As a result we are happy and life goes on.

Reports are nice! People are nice! So we decide to automate the Tribble line. (Trekkies, please forgive me!) We analyze our 5 options intelligently again and decide to purchase the Klingon Package.

Then it happens. People begin to complain. The CPU is too busy! The disc I/O is imbalanced! There are too many I/O requests against the database manager! The I/O bandwidth is not wide enough! Now be truthful. How many of your users call and make these statements? They probably call and say, "The system seems to be running slow." If they are EDP knowledgeable, they may politely ask, "Is the system down?"

In this day and age we purchase tools to solve our problems. It may be new applications for our users or new tools for EDP. We purchase database managers, performance monitors, editor tools for programmers, system manager tools, and electronic messaging systems. We are tool happy, and I think it's great. I have used many of them. I added several CPU's to my data center just like you. I implemented new applications just like you. I added more tools just like you. I solved MIS management problems just like you. And just like you, I seldom sat back and asked myself an important question or two. Are these decisions making my company more profitable, or just increasing costs? Are we becoming winners in our competitive marketplace?

Before we begin to analyze another vital approach to measuring our MIS EDP decisions, both short and long term, I would like to use an analogy that is easy to relate to and comprehend.

I will propose my new tool to you, my possible solution, that is simple but accurate. It's easy. It's logical. It's measurable. It's a "stopwatch." Now that I have you hanging, let's look at my analogy.

Let's look at A.J. Foyt and the Indianapolis 500. I think we can all agree that A.J. is a winner. But when you are a real winner, you count, you measure, you plan, you work hard and in the end you win. He has won 4 times in the many starts he has had.

So let's look at how A.J. Foyt might measure his performance. First he has feel for his equipment just like we do. He knows if it will run well or if it will run slow — but feelings don't count in the world of winners. So he times his laps very carefully and discovers he went 203 miles per hour. That sounds pretty good, but for the last few years the winning lap times have been around 216 miles per hour, so he takes the car to his mechanic (system manager). He hooks it up to his engine analyzer and tunes up the engine with a little tweaking (cache control on or off). Or perhaps he decides a major carburetor overhaul is needed (or adding four more megs or a 68-70 upgrade with more memory cache). Our analyzer, whether it is OPT3000 or Sysview or even good old SOO, says this machine is HOT. This hummer is going to fly!!

So A.J. takes his hot system out to the track. A.J. is a winner. He knows it's not one thing like a tuneup, but the total system (man, machine, track, everything), so he checks out his hot winner. How? With that "electronic stopwatch." Now that hot car goes 201.956 miles per hour. Slower!

Have you ever tuned your system? Have you ever turned on or off one of the disc's caching? Have you ever moved a file? Have you ever reloaded a database, or hopefully used a time-saving tool like MPEX or DBGeneral or Adager, and then from the user you hear the dreaded statement,"God, what did you do? Its even slower!" As a winner we also need to go back to the simple, basic measurements and goals that we based our system decisions on. We wanted to save our users time. We don't save CPU seconds, or disc I/0's! I do agree they may lead to the end result, just like a carburetor adjustment helped A.J. run faster. The reason we bought the system or application was to let our user do more things faster!

As you add AP applications you can't cause manufacturing to go from 3.2 second to 4.2 second response time on a basic lot move transaction. I know you say it's just one second! But we can't forget that we do 36,000 lot move transactions every month. That's 600 minutes. That is 10 man hours each and every month. That might allow us to produce two more wafers each month. With the 40% yield we get on our wafers that equates to 5000 chips each month. Since they are $5 each, that one little second just cost us $25,000 in lost revenue each month! Remember, just one second per transaction and you can lose $25,000 in revenue each month.

So now let's get to work with our "electronic stopwatch." As I visit sites and talk to people, I ask "What do you do on your system?" The usual response is, "Everything!" But after some other questions, it gets down to the following:

"On system one, our Series 70, is our manufacturing system."

"On system two, our second 70, is order processing."

"And on system three, a 48, is our accounting, general ledger, and payroll."

Frequently they are packages like ASK Manman, MM3000, IC-10, or SFD, but the key is each system has a primary function. When we bought the hardware and software, we determined what kinds of information we wanted to collect and how we were going to collect it. We figured out we would do X,000 lot move transactions each month. We also decided who would the input in the wafer fab area. We have decided the 2-3 seconds to do it was worth it for the management information we were going to get. Why do we then stop monitoring our management decision objectives?

Of course we all want to become more efficient, so we add more applications like HPDesk to help our engineers become more productive. That kind of productivity is certainly a tough item to measure! But can we measure its impact on production to determine what is the cost to that environment? Not just software cost or cost for added hardware, but what are the human costs in the fab production areas? Remember our 1 second delay from our earlier example? What we need to monitor and track, both on-line and historically, is the major goals and objectives we set when we bought the system. What is our production throughput?

Our application package tells us how much product throughput we achieve like wafers per month/week/or hour. Our lot tracking system tells us which wafer lot was produced on what day with what materials. But we don't track these critical items in EDP for some reason. How many lot move transactions do we do each hour? Are morning workloads heavier than the afternoon workloads? Who is on the system now and how many transactions are they doing per hour? How fast is the system responding to them? Are 90% of the transactions still under the target response time we as managers set when we bought the system? If we purchase application packages to answer management questions about those critical areas, doesn't it seem to make sense to do the same in EDP?

Now I know as an MIS manager we all say, "But how can I fix my problems if I can't determine what is wrong?" I wholeheartedly agree. I used OPT3000 as an MIS manager and I used it a lot. I can't imagine A.J. being a winner without using his engine analyzer. But he doesn't stop there, so why should you!

I remember once as a young MIS manager I needed another system. I had been to many staff meetings and watched the manufacturing management team convince the Division Manager to buy another $100,000 piece of fab equipment.

They had all their facts together regarding throughput, operational impact, people impact, labor vs. equipment tradeoffs, and all those ratios. So when I felt it was time to get my next CPU on order, I got my staff together. I put all the facts together to show my 48 was I/O bound because each GIC was doing 20-25 I/O's per second. I had graphs showing the

CPU was 80% to 85% busy between 9 and 12 in the morning and 1 and 4 in the afternoon. I had proof that I couldn't add any more terminals to my system. I was ready.

After an effective presentation, the marketing and manufacturing managers even reinforced the fact that the system was running slower. I felt for sure the Division Manager would sign the order for a new Series 68. Instead he asked questions like: "How *much* slower is the system? Can we move some work to the noon-time window when the CPU is only 40% busy? We get new fab equipment when it is fully loaded; what can we do to get the CPU 100% busy?"

Does all that sound familiar to you? Remember, we didn't get the system with objectives oriented around these goals. I realized right then I needed ACCURATE APPLICATION DATA TO SHOW WE WERE OVERLOADING OUR SYSTEM. I do want to emphasize measurement. We use the system for production purposes, even if it's an office automation machine. HPDesk is our production application when that is the situation. Our managers don't want to wait five seconds after each return key any more than our production people do. But in order to manage those systems, we have to know the basic response times of the transactions, the transaction throughput, and the amount of CPU resources needed to process each transaction. These items measure the true system performance, the man/machine interfaced system that truly drives our production environment.

Incidentally, we have had our electronic stopwatch running for the last hour. And now suddenly Shirley the user calls. She says performance has become slow. We decide to check the status of the terminals and we see she is on port 45 right now. We then request terminal timing, on-line, for the production users. We see Shirley has been on the system since 4:08 and has done 28 transactions. Nothing looks abnormal, so we will decide to watch her for a while.

We can see the form she is using in VPLUS, the program she is running, and the detailed response times she is getting from the system. We certainly can observe if any transactions are starting to take longer. The nice thing is we are only monitoring the critical production work. We are classifying by time, production program, VPLUS form (if used), and I/O port. These transactions and their pertinent status are logged and stored to a data base. Besides looking at a specific port, we should be able to request information based on the production program and its forms file, if its usage is collected.

Before we take our theory any further, let's look a little closer at our clock interface. It tells us what kind of time our system is using. For example, here we have a response time of only 1.7 seconds on each transaction, while our user has spent 3.48 seconds on the average to fill in the data. We should also notice our program takes 1.11 seconds, what our VPLUS overhead is and our stopwatch elapsed time. We also can determine that 3 of the 77 inputs were over 5 seconds, and by continued monitoring we can see if the count of long responses starts climbing. We have good "what's happening now" information.

The real key to our data is not the current information. That's important, but more important is the relationships between the present and our historical information. What is the trend of our usage? Now I do agree there are trend servers available. I also question whether they show us the vital data we need to make our management decisions.

The fact that I averaged 10 I/O's per disc and was 60% CPU busy two months ago is important. And the fact that I am 80% CPU busy with 15 I/O's per second per disc is also valid data. But nowhere in the analysis is the fact I added TurboIMAGE, and disc caching, and that I now do 5000 more transactions per month. Also not related is the information that response time has improved because of those additions. What I want to know, after adding disc caching, is how many more transactions can I accomplish per hour and what is the response time impact to my user. My user only measures me by my ability to provide adequate services for him to get his job done.

What can our electronic stopwatch tell us? Let's look at some theoretical examples. We are happily charging along on our 68 with no disc caching and 4 megabytes of memory. Users are slightly disgruntled because response time is getting sluggish. As MIS managers, we decide we need to do something because the analysis of our response times has shown us that 5% of our transactions are now over the 3 second response threshold we had set.

Six months ago when we upgraded from a 48 to a 68, we had a lot of problems. About 40% of our transactions were over 3 seconds and 10% were over 5 seconds. We were doing 60,000 transactions per month and everyone was complaining. After the upgrade everyone was happy.

So we set our response time threshold to 3 seconds. In those intervening 6 months our volume has grown at about 5000 transactions per month to the current 90,000 per month. And as we said earlier it has slowly degraded.

So what do we do? Well, the CPU has become busier according to our analyzer, but based on our experience with the 48, that 65% CPU busy is still okay. No discs or I/O channels show any signs of reaching bandwidth capacity, our memory manager has been doing only a little bit more memory swapping and disc caching hasn't increased at all. Now remember, we have had both our engine analyzer and our electronic timer at work!

A bit down the road (we all know about delivery schedules) we have our hardware, software, firmware, SE, and CE all staged for our exciting installation! We install and run some small tests and decide that everything looks good. The next day our friendly users show up and instead of our 2 session test, it's back to the real world of 70 sessions.

Within one hour, the phone is ringing off the hook. "Yee gads, what did you do? Install a turtle for our squirrel power?" Since I know the answer to this problem, I am now going to take my author's privilege! I am not going to analyze the problem as I did, stopwatch first, but the way you probably would! You break out your friendly engine analyzer. It runs as promised and shows us the CPU isn't too busy, the I/O channels aren't at a limit, the busiest disc is only at 15 I/O's per second and the GIC is at 20 per second. The memory has so much space, it's barely doing any swapping (once in 10 minutes), and we are befuddled. I log onto HPDesk and I am flying like crazy. What's wrong? After a while we go down to the manufacturing user to see if he is telling the truth. Sure enough his response time is a mess! So we turn caching off on all the drives. Now the user is seeing a response time about where we were *before* the upgrade.

Well, the problem was not memory and disc caching. It was an IMAGE problem. After we turned on TurboIMAGE things really took off and disc caching helped. The key to this story is the fact that memory and disc caching, and the money we put into them, was not the problem.

The point is, the reason to buy the hardware solution was to improve production throughput. After we installed our engine analyzer, we said "wow,we sure are running smoother." But like A.J., we didn't go faster on the track. Our stopwatch told us we were running slower like him. After careful analysis, we discovered IMAGE to be our problem, and happily TurboIMAGE solved it. At least, the other divisions using the same software would not make the same mistake. This example has shown we can use our management tools to troubleshoot our problems and really measure true system performance impact.

What if the upgrade had only added .4 second response time for our users in the fab? Would they have complained? Would we have known without our stopwatch? Remember our earlier 1 second on 36,000 transactions? We are now talking .4 second on 90,000 transactions, or 36,000 seconds again. Right back to that $25,000 potential revenue loss.

Again, important to my job as an MIS manager was my planning role. The electronic stopwatch had helped enormously in tuning and preparing of my systems. Since I know how many transactions I am doing and the total CPU needs of the applications, I could better balance my system loads by analyzing which user uses how much in resources. I know who to move from one system to another. I had at that point gone from a 48 shop to a shop with two 68's but I found a new challenge in capacity planning.

By this time, things had changed; the 70's had been announced and my division manager had asked me a simple question: Should we get another CPU or system or would two 70 upgrades be better?

Since we had been collecting transaction information for some time, we had good information on how many transactions were done by each individual, or process. Based on production profiles, if the fab wanted to increase production by 15%, we knew how many transactions, what type, and the typical response times. We also knew the average CPU seconds needed per transaction.

This information was the key to capacity planning. If the fab had to get 15% more IC die in a given area, it meant certain wafers had to be produced in a higher quantity. Manufacturing was good at accurately predicting what additional equipment would be needed since they have unit capacity for lappers, furnaces, and photo-masking equipment. What was nice was EDP had the same tools at last. We knew by wafer type what type of transactions were needed to produce the products. We also knew the average resources EDP needed to process those added transactions.

If we had 10,000 new transactions that in the past had used five CPU seconds per transaction, we knew we had to come up with 50,000 CPU seconds in additional capacity. In addition our job controller history statistics helped us track job demands and forecast job-related needs. Armed with CPU second needs we could do some ratios to determine whether two 70's could manage our future load. Data center accounting had shown us ratios between MIPS and CPU seconds available on our 48's and 68's. By extrapolation we could determine the number of potential CPU seconds on a new 70.

For example, a 50 second job on a .75 MIP machine might only need 30 seconds on a 1.4 MIP machine. These ratios helped us determine what resource would be needed by product line, department, and CPU. Knowledge of application interdependencies from our job controllers and application specialists helped us determine which jobs, screens, and

applications could best be split up. The net result was an enjoyable justification trip to the Division Manager.

The growth in Manufacturing and Order Processing for the next 12 months would approach the capacity limits of the two new 70's, if we got them. Since we could explain MIPS and how it determined available seconds, along with the additional needed capacity and planned response time, the Division Manager had no problem buying our third 68. In addition, HPDesk was in the midst of implementation, and based on its projected resource growth he signed off on one Series 70 upgrade. The second and third upgrade was planned for the next year.

The key to giving us this power for capacity planning was our electronic stopwatch. Transaction level resources were available as a result. User-acceptable goals could be set and maintained since they were monitorable. It was important that our division team had management-related goals to measure our equipment in EDP against. They could relate to our needs without needing to understand our technical jargon. We in EDP could continue to be concerned with "tuning our engine" just like the process engineers in wafer fab tune theirs.

In summary, I hope I have stimulated you to look at the world of EDP management/system performance measurement in a new light. Like our high-tech manufacturing counterparts, we must learn to measure and manage our unique technology in a way that managers who make the Profit/Loss decisions can understand.

We at Unison have dedicated ourselves to those Data Center Management tools you need to accomplish that. If you have problem areas feel free to give us a call. It isn't often the professional at the other end of the line has MIS Management experience comparable to yours, but at Unison that's the rule, not the exception.

#